DB2 10 for z/OS

Managing Performance

SC19-2978-17

DB2 10 for z/OS

Managing Performance

IBM

SC19-2978-17

Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

October 10, 2017 edition

This edition applies to DB2 10 for z/OS (product number 5605-DB2), DB2 10 for z/OS Value Unit Edition (product number 5697-P31), and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

Contents

About this information

This information describes performance management tasks for DB2® 10

Throughout this information, “DB2” means “DB2 10 for z/OS®”. References to other DB2 products use complete names or specific abbreviations.

Important: To find the most up to date content, always use IBM® Knowledge Center, which is continually updated as soon as changes are ready. PDF manuals are updated only when new editions are published, on an infrequent basis.

This information assumes that your DB2 subsystem is running in DB2 10 new-function mode.

Availability of new function in DB2 10

Generally, new SQL capabilities, including changes to existing functions, statements, and limits, become available only in new-function mode, unless explicitly stated otherwise. Exceptions to this general statement include optimization and virtual storage enhancements, which are also available in conversion mode unless stated otherwise. In DB2 Version 8 andDB2 9, most utility functions were available in conversion mode. However, for DB2 10, most utility functions become available in new-function mode.

Who should read this information

This information is primarily intended for system and database administrators.

It assumes that the user is familiar with:

- The basic concepts and facilities of DB2
- Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF)
- The basic concepts of Structured Query Language (SQL)
- The basic concepts of Customer Information Control System (CICS®)
- The basic concepts of Information Management System (IMS™)
- How to define and allocate z/OS data sets using job control language (JCL).

Certain tasks require additional skills, such as knowledge of Transmission Control Protocol/Internet Protocol (TCP/IP) or Virtual Telecommunications Access Method (VTAM®) to set up communication between DB2 subsystems.

DB2 Utilities Suite for z/OS

Important: In DB2 10, the DB2 Utilities Suite for z/OS is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

DB2 Utilities Suite for z/OS can work with DB2 Sort for z/OS and the DFSORT program. You are licensed to use DFSORT in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- II14047/II14213: USE OF DFSORT BY DB2 UTILITIES

- II13495: HOW DFSORT TAKES ADVANTAGE OF 64-BIT REAL ARCHITECTURE

These informational APARs are periodically updated.

Related concepts:

DB2 utilities packaging (DB2 Utilities)

Terminology and citations

When referring to a DB2 product other than DB2 for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

DB2 Represents either the DB2 licensed program or a particular DB2 subsystem.

Tivoli® OMEGAMON® XE for DB2 Performance Expert on z/OS
Refers to any of the following products:
- IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS
- IBM Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS
- IBM DB2 Performance Expert for Multiplatforms and Workgroups
- IBM DB2 Buffer Pool Analyzer for z/OS

C, C++, and C language
Represent the C or C++ programming language.

CICS Represents CICS Transaction Server for z/OS.

IMS Represents the IMS Database Manager or IMS Transaction Manager.

MVS™ Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®
Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for DB2 10 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including DB2 10 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

Tip: The IBM Knowledge Center (which includes information for DB2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the DB2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for z/OS documentation.

Send your comments by email to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).

Part 1. Managing DB2 performance

Managing the performance of DB2 is an iterative process. Periodically, or after significant changes to your system or workload, you must reexamine your objectives, and refine your monitoring and tuning strategy accordingly.

About this task

You can avoid some performance problems completely by planning for performance when you first design your system. As you begin to plan your performance, remember the following information.

- DB2 is only a part of your overall system. Any change to the system hardware, disk subsystems, z/OS, IMS, CICS, TCP/IP, VTAM, the network, WebSphere®, or distributed application platforms (such as Windows, UNIX, or Linux) that share your enterprise IT infrastructure can affect how DB2 and its applications run.
- The recommendations for managing performance are based on current knowledge of DB2 performance for "normal" circumstances and "typical" systems. Therefore, you need to understand that this information might not necessarily be the best or appropriate advice for every specific situation. In particular, the advice on performance often approaches situations from a performance viewpoint only. Other factors of higher priority might make some of these performance recommendations inappropriate for your specific solution.
- The recommendations are general. Performance measurements are highly dependent on workload and system characteristics that are external to DB2.

Procedure

To manage DB2 performance, use the following approaches:

1. Establish your performance objectives when you plan your system.
2. Consider performance as you design and implement your system.
3. Plan how you will monitor performance and capture performance data.
4. Analyze performance reports to decide whether the objectives have been met.
5. If performance is thoroughly satisfactory, use one of the following options:
   - Monitor less, because monitoring itself uses resources.
   - Continue monitoring to generate a history of performance to compare with future results.
6. If performance has not been satisfactory, take the following actions:
   a. Determine the major constraints in the system.
   b. Decide where you can afford to make trade-offs and which resources can bear an additional load. Nearly all tuning involves trade-offs among system resources.
   c. Tune your system by adjusting its characteristics to improve performance.
   d. Continue to monitor the system.

Related concepts:

DB2 performance management (Introduction to DB2 for z/OS)

Related tasks:

Programming applications for performance

Related information:

- DB2 12 for z/OS Performance Topics (IBM Redbooks)
- DB2 11 for z/OS Performance Topics (IBM Redbooks)
- DB2 10 for z/OS Performance Topics (IBM Redbooks)

Chapter 1. Setting performance objectives and defining your workloads

Before installing DB2, you should gather design data and evaluate your performance objectives with that information.

About this task

Reasonable performance objectives are realistic, in line with your budget, understandable, and measurable. How you define good performance for your DB2 subsystem depends on your particular data processing needs and their priority.

Mutual agreements about acceptable performance, between the data processing and user groups in an organization, are often formalized and called *service level agreements*. Service-level agreements can include expectations of query response time, the workload throughput per day, hour, or minute, and windows provided for batch jobs (including utilities). These agreements list criteria for determining whether or not the system is performing adequately. For example a service-level agreement might require that 90% of all response times sampled on a local network in the prime shift be under 2 seconds, or that the average response time not exceed 6 seconds even during peak periods. (For a network of remote terminals, consider substantially higher response times.)

Procedure

To define the workload of the system:

1. Set your initial performance objectives. Typical objectives specify values for the following measurements:

   Acceptable response time
   A duration within which some percentage of all applications have completed.

   Average throughput
   The total number of transactions or queries that complete within a given time.

   System availability
   Which included mean time to failure and the durations of down time.

   Objectives such as these define the workload for the system and determine the requirements for resources, such as processor speed, amount of storage, additional software, and so on. Often, however, available resources limit the maximum acceptable workload, which requires revising the objectives.
2. Consider the amount of processing that is expected. You might define your criteria in terms of the average, the ninetieth percentile, or even the worst-case response time. Your choice can depend on your site's audit controls and the nature of the workloads.

   z/OS Workload Manager (WLM) can manage to the performance objectives in the service-level agreement and provide performance reporting analysis. The terms used in the service-level agreement and the WLM service policy are similar.
3. Determine the types of workloads. For each type of workload, describe a preliminary workload profile that includes the following information:

- A definition of the workload type in terms of its function and its volume. You are likely to have many workloads that perform the same general function (for example, order entry through CICS, IMS, WebSphere Application Server, or other transaction managers) and have an identifiable workload profile. Other workload types include SPUFI and QMF™ queries, transactions, utilities, and batch jobs.

  For the volume of a workload that is already processed by DB2, use the summary of its volumes from the DB2 statistics trace.
- The relative priority of the type, including periods during which the priorities change.
- The resources that are required to do the work, including physical resources that are managed by the operating system (such as real storage, disk I/O, and terminal I/O) and logical resources managed by the subsystem (such as control blocks and buffers).

What to do next

You should review and reevaluate your performance objectives at all phases of system development.

Related concepts:

MVS Planning: Workload Management

Related tasks:

Setting performance objectives for distributed workloads by using z/OS Workload Manager

Determining z/OS Workload Manager velocity goals

Setting workload management goals (DB2 Data Sharing Planning and Administration)

Sizing your workloads

You can look at base estimates for transactions, query use, and batch processing to find ways to reduce the workload.

About this task

Changes in design in early stages, before contention with other programs, are likely to be the most effective. Later, you can compare the actual production profile against the base. Make an estimate even if these quantities are uncertain at this stage.

Procedure

To establish your resource requirements:

Estimate resource requirements for the following items:

Transactions

- Availability of transaction managers, such as IMS, CICS, or WebSphere
- Number of message pairs for each user function, either to and from a terminal or to and from a distributed application
- Network bandwidth and network device latencies

- Average and maximum number of concurrent users, either terminal operators or distributed application requesters
- Maximum rate of workloads per second, minute, hour, day, or week
- Number of disk I/O operations per user workload
- Average and maximum processor usage per workload type and total workload
- Size of tables
- Effects of objectives on operations and system programming

Query use

- Time required to key in user data
- Online query processing load
- Limits to be set for the query environment or preformatted queries
- Size of tables
- Effects of objectives on operations and system programming

Batch processing

- Batch windows for data reorganization, utilities, data definition activities, and BIND processing
- Batch processing load
- Length of batch window
- Number of records to process, data reorganization activity, use of utilities, and data definition activity
- Size of tables and complexity of the queries
- Effects of objectives on operations and system programming

Translating resource requirements into performance objectives

Your estimated resource requirements are in important input into the process of defining performance objectives.

Procedure

To translate your resource requirements into performance objectives:

1. For each workload type, convert your estimated resource requirements into measurable performance objectives. Include the following factors when you consider your estimates:

   System response time

   You cannot guarantee requested response times before any of the design has been done. Therefore, plan to review your performance targets when you design and implement the system.

   Response times can vary for many reasons. Therefore, include acceptable tolerances in your descriptions of targets. Remember that distributed data processing adds overhead at both the local and remote locations.

   Exclude from the targets any unusual applications that have exceptionally heavy requirements for processing or database access, or establish individual targets for those applications.

   Network response time

   Responses in the processor are likely to be in microseconds, whereas responses in the network with appropriate facilities can be about a millisecond. This difference in response times means that an overloaded network can impact the delivery of server responses to user terminals or distributed applications regardless of the speed of the processor.

Disk response time
I/O operations are generally responsible for much internal processing time. Consider all I/O operations that affect a workload.

Existing workload.
Consider the effects of additional work on existing applications. In planning the capacity of the system, consider the total load on each major resource, not just the load for the new application.

Business factors
When calculating performance estimates, concentrate on the expected peak throughput rate. Allow for daily peaks (for example, after receipt of mail), weekly peaks (for example, a Monday peak after weekend mail), and seasonal peaks as appropriate to the business. Also allow for peaks of work after planned interruptions, such as preventive maintenance periods and public holidays. Remember that the availability of input data is one of the constraints on throughput.

2. Include statements about the throughput rates to be supported (including any peak periods) and the internal response time profiles to be achieved.
3. Make assumptions about I/O rates, paging rates, and workloads.

Reviewing performance during external design

You should review performance during the external design phase for your system.

Procedure

During the external design phase, you must:

1. Estimate the network, web server, application server, processor, and disk subsystem workload.
2. Refine your estimates of logical disk accesses. Ignore physical accesses at this stage. One of the major difficulties is determining the number of I/Os per statement.

Reviewing performance during internal design

You should review performance objectives during the internal design of your system.

Procedure

During the internal design phase, you must:

1. Refine your estimated workload against the actual workload.
2. Refine disk access estimates against database design. After internal design, you can define physical data accesses for application-oriented processes and estimate buffer hit ratios.
3. Add the accesses for DB2 work file database, DB2 log, program library, and DB2 sorts.
4. Consider whether additional processor loads can cause a significant constraint.
5. Refine estimates of processor usage.

6. Estimate the internal response time as the sum of processor time and synchronous I/O time or as asynchronous I/O time, whichever is larger.
7. Prototype your DB2 system. Before committing resources to writing code, you can create a small database, update the statistics stored in the DB2 catalog tables, run SQL statements and examine the results.

Reviewing performance during coding and testing

You should review your performance objectives during the coding and testing phase for your system.

Procedure

During the coding and testing phases, you must:

1. Refine the internal design estimates of disk and processing resources.
2. Run the monitoring tools you have selected and check the results against your estimates. You might use a terminal network simulator such as Teleprocessing Network Simulator (TPNS) or other tools to test the system and simulate load conditions.

Related tasks:

Testing DB2 performance

➡ Testing and debugging an application program on DB2 for z/OS (DB2 Application programming and SQL)

Related reference:

➡ What is Teleprocessing Network Simulator? (TPNS General Information)

Reviewing performance after development

When you are ready to test the complete system, review its performance in detail.

Procedure

Take the following steps to complete your performance review:

1. Validate system performance and response times against your performance objectives.
2. Identify resources whose usage requires regular monitoring.
3. Incorporate the observed figures into future estimates:
   a. Identify discrepancies from the estimated resource usage
   b. Identify the cause of the discrepancies
   c. Assign priorities to remedial actions
   d. Identify resources that are consistently heavily used
   e. Set up utilities to provide graphic representation of those resources
   f. Project the processor usage against the planned future system growth to ensure that adequate capacity is available
   g. Update the design document with the observed performance figures
   h. Modify your procedures for making estimates according to what you have learned what you have learned

Results

You need feedback from users and might have to solicit it. Establish reporting procedures, and teach your users how to use them. Consider logging incidents such as:

- System, line, and transaction or query failures
- System unavailable time
- Response times that are outside the specified limits
- Incidents that imply performance constraints, such as deadlocks, deadlock abends, and insufficient storage
- Situations, such as recoveries, that use additional system resources

You should log detailed information for such incidents.

- Time
- Date
- Location
- Duration
- Cause (if it can be determined)
- Action taken to resolve the problem

Chapter 2. Planning to review performance data

When establishing requirements and planning to monitor performance, you should also plan how to review the results of monitoring.

About this task

You can inspect your performance data to determine whether performance has been satisfactory, to identify problems, and to evaluate the monitoring process.

Procedure

- Plan to review the performance data systematically. Review daily data weekly and weekly data monthly; review data more often when reports raise specific questions that require investigation. Depending on your system, the weekly review might require about an hour, particularly after you have had some experience with the process and are able to locate quickly any items that require special attention. The monthly review might take half a day at first, less time later on. But when new applications are installed, workload volumes increased, or terminals added, allow more time for review.
- Review the data on a gross level, looking for problem areas. Review details only if a problem arises or if you need to verify measurements.
- When reviewing performance data, try to identify the basic pattern in the workload, and then identify variations of the pattern. After a certain period, discard most of the data you have collected, but keep a representative sample. For example, save the report from the last week of a month for three months; at the end of the year, discard all but the last week of each quarter. Similarly, keep a representative selection of daily and monthly figures. Because of the potential volume of data, consider using Tivoli Decision Support for z/OS, Application Monitor for z/OS, or a similar tool to track historical data in a manageable form.

Related concepts:

Planning for performance monitoring

Related tasks:

Investigating DB2 performance problems

Typical review questions

You can use specific review questions to help guide your review of performance data.

Use the following questions as a basis for your own checklist. They are not limited strictly to performance items, but your historical data can provide most of their answers. If the performance data is for modeled workloads or changed workloads, the first question to ask for each category is, "What changed?"

How often was each transaction and SQL statement used?

1. Considering variations in the workload mix over time, are the monitoring times appropriate?
2. Should monitoring be done more frequently during the day, week, or month to verify this?

3. How many SELECT, INSERT, UPDATE, DELETE, PREPARE, DESCRIBE, DESCRIBE TABLE, PREPARE, OPEN, FETCH, and CLOSE statements are issued per second and per commit?
4. How many IRLM and buffer pool requests are issued per second and per commit?

How were processor and I/O resources used?

1. Has usage increased for functions that run at a higher priority than DB2 tasks? Examine CICS, IMS, z/OS, JES, TCP/IP, VTAM, WebSphere Application Server, WebSphere MQ (formerly called MQSeries®), other subsystems, and key applications.
2. Is the report of processor usage consistent with previous observations?
3. Are scheduled batch jobs able to run successfully?
4. Do any incident reports show that the first invocation of a function takes much longer than later ones? This increased time can happen when programs have to open data sets.
5. What is the CPU time and where is it accumulated? Separate CPU time into accounting TCB and SRB time, and distinguish non-nested, stored procedure, user-defined function, and trigger CPU times. Note the times for DB2 address spaces, DBM1, MSTR, IRLM, and DDF.
6. In a data sharing environment, how are the coupling facility (CF) lock, group buffer pool, and SCA structures performing? What is the peak CF CPU utilization?
7. Are unnecessary DB2 traces on?
8. Are online monitors performing unnecessary or excessive tracing?

How much real storage was used, and how effective is storage?

1. Is the paging rate increasing? Adequate real storage is very important for DB2 performance.
2. What are the hit ratios for the following types of storage:
   - Buffer pools
   - EDM statement cache
   - EDM DBD cache
   - EDM skeleton pool

To what degree was disk used?

Is the number of I/O requests increasing? DB2 records both physical and logical requests. The number of physical I/Os depend on the configuration of indexes, the data records per control interval, and the buffer allocations.

To what extent were DB2 log resources used?

1. Is the log subject to undue contention from other data sets?

   Recommendation: Do not put a recoverable (updated) resource and a log under the same RAID controller. If that controller fails, you lose both the resource and the log, and you are unable to perform forward recovery.
2. What is the I/O rate for requests and physical blocks on the log?
3. What is the logging rate for one log in MB per second?
4. How fast are the disk units that are used for logging?

Do any figures indicate design, coding, or operational errors?

1. Are disk, I/O, log, or processor resources heavily used? If so, was that heavy use expected at design time? If not, can the heavy use be explained in terms of heavier use of workloads?
2. Is the heavy usage associated with a particular application? If so, is there evidence of planned growth or peak periods?
3. What are your needs for concurrent read/write and query activity?
4. Are there any disk, channel, or path problems?
5. Are there any abends or dumps?

What are the effects of DB2 locks?

1. What are the incidents of deadlocks and timeouts?
2. What percentage of elapsed time is due to lock suspensions? How much lock or latch contention was encountered? Check the contention rate per second by class.
3. How effective is lock avoidance?

Were there any bottlenecks?

1. Were any critical thresholds reached?
2. Are any resources approaching high utilization?

Related concepts:

Accounting trace

Statistics trace

Monitoring CICS, and IMS

Related tasks:

Monitoring concurrency and locks

Monitoring system resources by using RMF

Monitoring I/O and storage

Validating your performance objectives

After beginning to review and monitor performance, you need to find out if your objectives are reasonable.

About this task

You should consider questions about the objectives, such as:

- Are they achievable, given the available hardware?
- Are they based upon actual measurements of the workload?

Procedure

To measure performance against initial objectives and report the results to users:

1. Identify any systematic differences between the *internal response time*, the measured performance data, and the *external response time,* what the user sees.
2. If the measurements differ greatly from the estimates:
   - Revise response-time objectives for the application
   - Upgrade your system
   - Plan a reduced application workload.

If, however, the differences between internal and external response times are not too large, you can begin monitoring and tuning the entire system.

Related tasks:

Monitoring and analyzing DB2 performance data

Monitoring performance

Investigating DB2 performance problems

Part 2. Managing system performance

By considering performance as you design and configure your system, you can help to ensure better performance from DB2.

Chapter 3. z/OS performance options for DB2

You can set z/OS performance options for DB2 by using z/OS Workload Manager.

With Workload Manager (WLM), you define performance goals and assign a business importance to each goal. You define the goals for work in business terms, and the system decides how much resource, such as CPU and storage, should be given to the work to meet its goal.

WLM controls the dispatching priority based on the goals you supply. WLM raises or lowers the priority as needed to meet the specified goal. Thus, you do not need to fine-tune the exact priorities of every piece of work in the system and can focus instead on business objectives.

The three kinds of goals are:

Response time
How quickly you want the work to be processed.

Execution velocity
How fast the work should be run when ready, without being delayed for processor, storage, I/O access, and queue delay.

Discretionary
A category for low priority work for which you define no performance goals.

Response times are appropriate goals for user applications, such as QMF users running under the TSO address space goals, or users of CICS using the CICS workload goals. You can also set response time goals for distributed users.

For DB2 address spaces, velocity goals are more appropriate. A small amount of the work done in DB2 is counted toward this velocity goal. Most of the work done in DB2 applies to user goals.

Related tasks:

Setting performance objectives for distributed workloads by using z/OS Workload Manager

Determining z/OS Workload Manager velocity goals

Related reference:

MVS Planning: Workload Management (MVS Planning: Workload Management)

Determining z/OS Workload Manager velocity goals

With z/OS Workload Manager (WLM), work is assigned to a service class. For each service class, you define both the wanted performance goal and a business importance of meeting the stated goal.

About this task

Transactional work in a z/OS system can be defined to run with either transaction response time goals or can be run with velocity goals. Generally, transaction response time goals are recommended because they provide more management control. However, velocity goals work well in many installations for workloads such as CICS, IMS, and WebSphere.

The WLM service definition of workloads must be defined with knowledge of all of the workloads that run in the LPAR and the parallel sysplex.

Important: The recommendations for DB2 workloads might need to be adjusted so they are correctly incorporated into the overall WLM policy design.

Procedure

To account for DB2 address spaces in a WLM service definition, apply the following recommendations:

- Place the IRLMPROC address space in SYSSTC dispatching priority. This address space manages IRLM locks and latches. This address space must run at a high dispatching priority that provides little or no CPU delay.
- Place the following address spaces, which are critical for efficient system operation, in the same user-designed service class. This class must be defined with aggressive performance goals and a high importance:

  ***ssnm*MSTR**
  Contains the DB2 system monitor task and requires an aggressive WLM goal so it can monitor CPU stalls and virtual storage constraints.

  ***ssnm*DBM1**
  Manages DB2 threads and is used for system services such as page set opening. In data sharing environments, this address pace is critical for global operations such as P-lock negotiations, notifies, and global commands.

  The *ssnm*DIST and WLM-managed stored procedure address spaces
  Address spaces that run only the DB2 service tasks and work for DB2 that is not attributable to a single user. These address spaces typically place a minimal CPU load on the system. However, they do require minimal CPU delay to ensure good system-wide performance, and to avoid queuing of threads.

  DDF workloads, with their higher CPU demands, are controlled by the WLM service class definitions for the DDF enclave workloads. Similarly, the higher CPU demands for processing stored procedures, are controlled by the WLM service class definitions for the workloads that call the stored procedures.

  Important: The velocity goal of these address spaces must be high enough that new work and new connections can get started and placed into their own goal, or the goal of the caller.
- Specify a high *importance* for the DB2 workloads, typically importance 1. In every case, the service class with the DB2 address spaces must be set with an understanding of the overall WLM service definition in use. Use this information as a guide in setting the DB2 portion of the overall WLM policy.
- Take the following actions to ensure that the DB2 environment receives the CPU service that it requires to ensure a well running system:
  - Ensure that functions like TCP/IP and VTAM are defined in SYSSTC.
  - If necessary to meet performance objectives, it might be appropriate to designate certain service classes as CPU Critical. Doing so provides long-term

CPU protection of critical work. When you designate a service class as CPU critical, you ensure that less important work generally has a lower dispatch priority than work that is marked CPU critical. This protection can be valuable for work that is extremely CPU-sensitive. Generally, having an appropriately set goal for the DB2 service class is all that is required to ensure a well running system. However, CPU critical protection is available if needed.

- Set the *velocity goal* for the DB2 address spaces to be among the highest for the current LPAR.

  The intent is for the DB2 address spaces to be defined so that they experience little CPU delay. These address spaces must be defined with a WLM velocity goal.

  Typically, these address spaces are placed together into the same user-defined service class. This user-defined service class be does not need to be dedicated to these address spaces. Because they must be defined with a velocity goal, it is important to set an appropriately high goal set for this workload.

  Velocity goals are dependent on the overall LPAR logical CP configuration, and the amount of processor capacity allocated to the LPAR. LPARs that have fewer logical CPs can attain only lower velocities. Whereas, LPARs that have more logical CPs can obtain higher velocities.

Table 1. Recommended velocity goal based on the number of central processors per LPAR

Number of CPs defined for the LPAR	Recommended velocity goal
1-5	50-70
6-15	60-80
More than 15	70-90

- For LPARs that are constrained by a lack of CPU resources and have lower priority work that uses DB2, use *blocked workload support*. This support is most important in an environment where work that uses DB2 resources runs constrained at low priorities. The low priority constrained work might hold DB2 resources required by other, more important, workloads in the system or sysplex.

  For environments where low dispatch priority DB2 workloads run in periods of CPU constraint, it might prove beneficial to help blocked workloads more frequently. Changing the default blocked interval, (BLWLINTHD in IEAOPTxx), from the default value of 20 seconds to 5-6 seconds, might provide better overall system throughput at high utilizations. However, long-term reliance on blocked workload support to accommodate CPU saturated systems is not recommended.

Related concepts:

How DB2 assigns I/O priorities

The DB2 system monitor

Related tasks:

System-provided service classes (MVS Planning: Workload Management)

Organizing work into workloads and service classes (MVS Planning: Workload Management)

Related reference:

MVS Planning: Workload Management (MVS Planning: Workload Management)

Systems Programmer's Guide to: Workload Manager (IBM Redbooks)

Related information:

- Long-Term CPU Protection (z/OS MVS Planning: Workload Management)
- z/OS Availability: Blocked Workload Support

How DB2 assigns I/O priorities

DB2 informs z/OS about which address space's priority is to be associated with a particular I/O request.

WLM handles the management of the request from after that. The table below describes to which enclave or address space DB2 associates I/O read requests.

Table 2. How read I/O priority is determined

Request type	Synchronous reads	Prefetch reads
Local	Application's address space	Application's address space
DDF or Sysplex query parallelism (assistant only)	Enclave priority	Enclave priority

The table below describes to which enclave or address space DB2 associates I/O write requests.

Table 3. How write I/O priority is determined

Request type	Synchronous writes	Deferred writes
Local	Application's address space	ssnmDBM1 address space
DDF	DDF address space	ssnmDBM1 address space

Chapter 4. Managing I/O processing, response time, and throughput

To ensure that DB2 meets your goals for response time and throughput, you must manage how your system uses I/O processing.

Related concepts:

Planning the placement of DB2 data sets

Related tasks:

Configuring storage for performance

Tuning database buffer pools

Setting limits for system resource usage by using the resource limit facility

Controlling the number of I/O operations

You can improve the response time of your applications and queries by reducing the number of unnecessary I/O operations.

Related concepts:

Planning the placement of DB2 data sets

How DB2 assigns I/O priorities

Related tasks:

Managing the opening and closing of data sets

Estimating concurrent I/O requests

Monitoring I/O activity of data sets

Organizing tables by hash for fast access to individual rows

Chapter 22, “Improving performance for LOB data,” on page 273

Minimizing the use of real and virtual storage

Designing EDM storage space for performance

Choosing data page sizes

Read operations and prefetch I/O

DB2 uses prefetch I/O to read data in almost all cases, but uses synchronous I/O in certain cases.

When *synchronous read* is used, just one page is retrieved at a time. The unit of transfer for a synchronous read is one page per single I/O operation. DB2 uses prefetch mechanisms such as dynamic prefetch, sequential prefetch, and list prefetch, whenever possible, to avoid costly wait times from synchronous I/O.

Prefetch is a mechanism for reading a set of pages, usually 32, into the buffer pool with only one asynchronous I/O operation. Prefetch provides for substantial savings in both CPU and I/O costs. The maximum number of pages read by a single prefetch operation is determined by the size of the buffer pool that is used for the operation.

DB2 uses the following types of prefetch to avoid using costly synchronous read to access data, indexes, and LOBs:

Sequential prefetch

DB2 uses *sequential prefetch* for table scans and for sequential access to data in a multi-table segmented table space when index access is not available.

Dynamic prefetch

In *dynamic prefetch*, DB2 uses a sequential detection algorithm to detect whether pages are being read sequentially. DB2 tries to distinguish between clustered or sequential pages from random pages. DB2 uses multi-page asynchronous prefetch I/Os for the sequential pages, and synchronous I/Os for the random pages.

For example, if the cluster ratio for an index is 100% and a table is read in key-sequential order according to that index, all of the data pages are clustered and read sequentially. However, if the cluster ratio is somewhat less than 100%, then some of the pages are random and only those pages are read using synchronous I/Os. Dynamic prefetch works for both forward and backwards scans.

Because dynamic prefetch uses sequential detection, it is more adaptable to dynamically changing access patterns than sequential prefetch. DB2 uses dynamic prefetch in almost all situations, the main exception is for table space scans. Index scans always use dynamic prefetch.

List prefetch

DB2 uses *list prefetch* to read a set of data pages that is determined by a list of record identifiers (RIDs) from an index or from the DB2 log.DB2 also uses list prefetch to read non-consecutive index leaf pages which are determined from the non-leaf pages, and to read LOB pages which are determined from the LOB map. Unlike other types of prefetch, list prefetch does not use any kind of sequential detection. Instead,DB2 uses list prefetch in certain situations, such as the following examples:

- Reading leaf pages of a disorganized index.
- The optimizer chooses a list prefetch access path.
- Fast log apply operations.
- Incremental image copies.
- Access to fragmented LOB data.
- RUNSTATS table sampling.

DB2 uses the RID pool to process the RID list for list prefetch. The size of the RID pool is controlled by the value of the MAXRBLK subsystem parameter. If the RID pool is too small to contain the RID list processing for a list prefetch operation, a table space scan might be used instead.DB2 might use work files to continue processing the RID list if the size of the RID pool is too small. The use of work files for RID list processing is controlled by the value of the MAXTEMPS_RID subsystem parameter.

You can use the sequential steal threshold (VPSEQT) to protect randomly accessed pages in the buffer pool. It is beneficial to protect the random pages from the sequential pages, because it is generally faster to read sequential pages than random pages from disk, and sequential pages are less likely to be re-accessed.

Because all prefetch I/Os are executed under a service request block in the DBM1 address space, the I/O time for prefetch I/Os is asynchronous with respect class 2 CPU time. When a get page operation waits for a prefetch I/O to complete, the class 3 suspension time is captured as "other read I/O" suspension.

Prefetch CPU time is captured as system SRB time. CPU time for prefetch is usually small, but it can become significant for index scans because the compressed index pages are decompressed by the prefetch engine.

The number of pages read by prefetch

The following table shows the number pages read by prefetch for each asynchronous I/O for each buffer pool size (4 KB, 8 KB, 16 KB, and 32 KB).

Table 4. The number of pages read for each asynchronous I/O by prefetch, by buffer pool size

Buffer pool size	Number of buffers	Pages Read by Sequential and LOB List Prefetch	Pages Read by Dynamic and Non-LOB list Prefetch	Pages Read by Utility sequential Prefetch
4 KB	VPSIZE < 224	8	8	16
	225 < VPSIZE <1,000	16	16	32
	1000 <= VPSIZE < 40,000 or VPSIZE*VPSEQT < 40000	32	32	64
	40,000 <= VPSIZE*VPSEQT < 80,000	64	32	64
	80,000 <= VPSIZE*VPSEQT	64	32	128
8 KB	VPSIZE < 48	4	4	8
	48 < VPSIZE <400®	8	8	16
	400 <= VPSIZE< 20,000 or VPSIZE*VPSEQT < 20000	16	16	32
	20,000 <= VPSIZE*VPSEQT < 40,000	32	16	32
	40,000 <= VPSIZE*VPSEQT	32	16	64
16 KB	VPSIZE < 24	2	2	4
	24 < VPSIZE < 200	4	4	8
	200 <= VPSIZE< 10,000 or VPSIZE*VPSEQT < 10000	8	8	16
	10,000 <= VPSIZE*VPSEQT < 20,000	16	8	16
	20,000 <= VPSIZE*VPSEQT	16	8	32
32 KB	VPSIZE < 12	1	1	2
	12 < VPSIZE < 100	2	2	4
	100 <= VPSIZE< 5,000 or VPSIZE*VPSEQT < 5,000	4	4	8
	5,000 <= VPSIZE*VPSEQT < 10,000	8	4	8
	10,000 <= VPSIZE*VPSEQT	8	4	16

Related concepts:

Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U')

Additional statistics that provide index costs

Guidelines for setting buffer pool thresholds

Related tasks:

Managing RID pool size

Related reference:

Buffer pool thresholds that you can change

RID POOL SIZE field (MAXRBLK subsystem parameter) (DB2 Installation and Migration)

MAX TEMP RID field (MAXTEMPS_RID subsystem parameter) (DB2 Installation and Migration)

Related information:

DB2 for z/OS and List Prefetch Optimizer (IBM Redbooks)

Write operations

Write operations are usually performed concurrently with user requests.

Updated pages are queued by data set until they are written when one of the following events occurs:

- A checkpoint is taken
- The percentage of updated pages in a buffer pool for a single data set exceeds a preset limit called the vertical deferred write threshold (VDWQT)
- The percentage of unavailable pages in a buffer pool exceeds a preset limit called the deferred write threshold (DWQT)

The following table lists how many pages DB2 can write in a single I/O operation.

Table 5. Number of pages that DB2 can write in a single I/O operation

Page size	Number of pages
4 KB	32
8 KB	16
16 KB	8
32 KB	4

The following table lists how many pages DB2 can write in a single utility I/O operation. If the number of buffers is large enough, DB2 can write twice as many pages for each I/O operation for a utility write.

Table 6. The number of pages that DB2 can write for a single I/O operation for utility writes

Page Size	Number of buffers	Number of pages
4 KB	BP ≥ 80,000	128
	BP < 80,000	64
8 KB	BP ≥ 40,000	64
	BP < 40,000	32
16 KB	BP ≥ 20,000	32
	BP < 20,000	16
32 KB	BP ≥ 10,000	16
	BP < 10,000	8

As with utility write operations, DB2 can write twice as many pages for each I/O in a LOB write operation. The following table shows the number of pages that DB2 can write for each I/O operation for a LOB write.

Table 7. The number of pages that DB2 can write for in a single I/O operation for LOB writes

Page Size	Number of buffers	Number of pages
4 KB	BP ≥ 80,000	64
	BP < 80,000	32
8 KB	BP ≥ 40,000	32
	BP < 40,000	16
16 KB	BP ≥ 20,000	16
	BP < 20,000	8
32 KB	BP ≥ 10,000	8
	BP < 10,000	4

Related concepts:

Buffer pool thresholds

Related reference:

Buffer pool thresholds that you can change

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Making buffer pools large enough for the workload

You might improve the performance of I/O operations by increasing the size of your buffer pools.

You should make buffer pools as large as you can afford for the following reasons:

- Using larger buffer pools might mean fewer I/O operations and therefore faster access to your data.
- Using larger buffer pools can reduce I/O contention for the most frequently used tables and indexes.
- Using larger buffer pools can speed sorting by reducing I/O contention for work files.

However, many factors affect how you determine the number of buffer pools to have and how big they should be.

Related tasks:

Choosing buffer pool sizes

Enabling automatic buffer pool size management

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

DSNTIP1: Buffer pool sizes panel 1 (DB2 Installation and Migration)

Making I/O operations faster

You can use several methods to reduce the time required to perform individual I/O operations.

Related concepts:

Parallel processing

Distributing data sets efficiently

Avoid I/O contention and increase throughput through the I/O subsystem by placing frequently used data sets on fast disk devices and by distributing I/O activity.

About this task

Distributing I/O activity is less important when you use disk devices with parallel access volumes (PAV) support and multiple allegiance support.

Putting frequently used data sets on fast devices

You can improve performance by assigning frequently used data sets to faster disk devices.

Procedure

To make the best use of your disk devices:

- Assign the most frequently used data sets to the faster disk devices at your disposal.
- For partitioned table spaces, you might choose to have some partitions on faster devices than other partitions. Placing frequently used data sets on fast disk devices also improves performance for nonpartitioned table spaces.
- Consider partitioning any nonpartitioned table spaces that have excessive I/O contention at the data set level.

Distributing the I/O

By distributing your data sets, you can prevent I/O requests from being queued in z/OS.

Procedure

To distribute I/O operations:

- If you do not have parallel access volumes, allocate frequently used data sets or partitions across your available disk volumes so that I/O operations are distributed. Even with RAID devices, in which the data set is spread across the physical disks in an array, data should be accessed at the same time on separate logical volumes to reduce the chance of an I/O request being queued in z/OS.
- Consider isolating data sets that have characteristics that do not complement other data sets.

Partitioning schemes and data clustering for partitioned table spaces:

Depending on the type of operations that your applications emphasize, you have several options for distributing I/O.

If the partitions of your partitioned table spaces must be of relatively the same size (which can be a great benefit for query parallelism), consider using a ROWID column as all or part of the partitioning key.

For partitions that are of such unequal size that performance is negatively affected, alter the limit key values to set new partition boundaries and then reorganize the affected partitions to rebalance the data. Alternatively, you can use the REORG utility with the REBALANCE keyword to set new partition boundaries.

REBALANCE causes DB2 to change the limit key values such that the rows in the range of partitions being reorganized are distributed across those partitions as evenly as possible.

If your performance objectives emphasize inserts, distribute the data in a manner that reduces the amount of clustered key values. Consider designing your database with randomized index keys design to remove clustering. You can also take advantage of the index page splitting by choosing the appropriate size for index pages. If the rate of inserts does not require you to spread out the inserts, consider creating or altering tables with the APPEND YES option.

In contrast, for performance objectives that emphasize read operations, your data clustering should reflect the sequence in which queries will be processed so that DB2 can use the sequential processing method of parallelism to reduce I/O and CPU time.

Partition data that will be subject to frequent update operations in a manner that provides plenty of free space, especially if new and updated rows might expand because they contain columns with varying-length data types or compressed data.

Related concepts:

Index splitting for sequential INSERT activity

Methods of parallel processing

Related tasks:

Designing databases for concurrency

Increasing the number of data sets for an index:

By increasing the number of data sets that are used for an index and spreading those data sets across the available I/O paths, you can reduce the physical contention on the index.

Using *data-partitioned secondary indexes*, or making the piece size of a nonpartitioned index smaller, increases the number of data sets that are used for the index.

A secondary index on a partitioned table space can be partitioned. When you partition an index, the partitioning scheme is that of the data in the underlying table, and each index partition has its own data set. Although data-partitioned secondary indexes promote partition independence and can provide performance advantages, they do not always improve query performance. Before using a data-partitioned secondary index, understand the advantages and disadvantages.

Creating additional work file table spaces to reduce contention

You can minimize I/O contention in certain situations by creating additional work file table spaces.

About this task

For a single query, the recommended number of work file disk volumes to have is one-fifth the maximum number of data partitions, with 5 as a minimum and 50 as a maximum. For concurrently running queries, multiply this value by the number of concurrent queries. Depending on the number of table spaces and the amount of concurrent activity, performance can vary. In general, adding more table spaces improves performance.

Procedure

To fine tune the combination of different types of tables in the work file database:

- In query parallelism environments, ensure that the number of work file disk volumes is at least equal to the maximum number of parallel operation use for queries in a given workload, place these volumes on different channel or control unit paths, and monitor the I/O activity for the work file table spaces.
  1. Place the volumes on different channel or control unit paths.
  2. Monitor the I/O activity for the work file table spaces. You might need to further separate this work file activity to avoid contention.
  3. As the amount of work file activity increases, consider increasing the size of the buffer pool for work files to support concurrent activities more efficiently. The general recommendation for the work file buffer pool is to increase the size to tune the following buffer pool statistics:
     - `MERGE PASSES DEGRADED`, which should be less than 1% of `MERGE PASS REQUESTED`.
     - `WORKFILE REQUESTS REJECTED`, which should be less than 1% of `WORKFILE REQUEST ALL MERGE PASSES`.
     - Synchronous read I/O, which should be less than 1% of pages read by prefetch
     - The sequential prefetch quantity, which should be:
       - 8 for 4 KB work file buffer pools
       - 2 for 32KB work file buffer pools

       You can calculate the sequential prefetch quantity by dividing the number of pages read by sequential prefetch (QBSTSPP) by the number of sequential prefetch reads (QBSTPIO).
- If your applications require extensive use of temporary objects or operations that can be processed in work files that span more than one table space, define multiple table spaces in the work file database that have the following preferred attributes:
  - Zero secondary quantity. (SECQTY= 0)
  - Stored in DB2-managed data sets.
  - Segmented (non-universal) table spaces.

  For processing that can span more than one table space, DB2 prefers to allocate space for work files in table spaces that have the preferred attributes. By creating multiple work file table spaces that have these attributes, you can support efficient concurrent read and write I/Os to the work files.

  When a table space in the work file database is stored in user-managed data sets, DB2 does not detect whether any secondary allocation exists. So, such table spaces are given the same preference as table spaces that have the preferred attributes, even when the table space has a secondary allocation.

  Processing that uses work files and can span more than one table space includes objects and operations such as:
  - Large concurrent sorts and single large sorts
  - Created temporary tables
  - Some merge, star, and outer joins
  - Non-correlated subqueries
  - Materialized views
  - Materialized nested table expressions
  - Triggers with transition variables

- If your applications require extensive use of temporary objects or operations that can be processed only in a single table space, define some table spaces in the work file database that have the following preferred attributes:
  - Partition-by-growth (regardless of the SECQTY value) or segmented table spaces that are not partitioned and have a non-zero SECQTY value.
  - Stored in DB2-managed data sets.

  DB2 gives preference for processing that cannot span more than one table space to table spaces that have the preferred attributes.The table spaces that have the preferred attributes help to minimize contention for space between temporary objects or operations that can span multiple tables spaces, and those that cannot.

  Processing that uses work files and is limited to a single table space includes objects and operations such as:
  - Declared global temporary tables
  - Scrollable cursors
  - SQL MERGE statements
- Ensure that the work file database contains at least one 32-KB page size table space before you create declared global temporary tables. The rows of declared global temporary tables reside in a table space in the work file database, and DB2 does not create an implicit table space for the declared global temporary table.
- To further improve performance, consider allocating more 32-KB data sets. DB2 uses 32 KB work file data sets when the total work file record size, including record overhead, exceeds 100 bytes, resulting in better performance and reduced work file buffer pool and disk space usage for work files, in some cases.

Example

GUPI

To create new work file table spaces:

1. Use the VSAM DEFINE CLUSTER statement to define the required data sets. You can use the definitions in the edited DSNTIJTM installation job as a model.
2. Create the work file table space by entering the following SQL statement (If you are using DB2-managed data sets, omit this step.):

   ```
   CREATE TABLESPACE xyz IN DSNDB07
      BUFFERPOOL BP7
      CLOSE NO
      USING VCAT DSNC100;
   ```

GUPI

Related concepts:

How sort work files are allocated

Segmented (non-UTS) table spaces (deprecated) (Introduction to DB2 for z/OS)

Virtual storage requirements for storage pools and working storage (DB2 Installation and Migration)

Secondary space allocation (DB2 Administration Guide)

Related tasks:

Defining data sets (DB2 Administration Guide)

Related reference:

SECONDARY QTY field (SECQTY subsystem parameter) (DB2 Installation and Migration)

CREATE TABLESPACE (DB2 SQL)

Statistics Report Buffer Pool Sort/Merge (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

Work file sizing (DB2 9 for z/OS Performance Topics)

Improving space allocation and pre-formatting

You can improve the performance of applications that use heavy insert processing by controlling how space is allocated and pre-formatted.

About this task

When inserting records, DB2 pre-formats space within a page set as needed. The allocation amount, which is either by *cylinder* or *track*, determines the amount of space that is pre-formatted at any one time.

Because less space is pre-formatted at one time for the track allocation amount, a mass insert can take longer when the allocation amount is track than the same insert when the allocation amount is cylinder. However, smart secondary space allocation minimizes the difference between track and cylinder allocation.

Cylinder allocation can reduce the time required to do SQL mass inserts and to perform LOGONLY recovery. It does not affect the time required to recover a table space from an image copy or to run the REBUILD utility.

Procedure

Use the following approaches to control space allocation and pre-formatting:

- Specify your space allocation amounts to ensure allocation by cylinder. The allocation amount depends on device type and the values that you specify for PRIQTY and SECQTY when you define table spaces and indexes. If you use record allocation for more than a cylinder, cylinder allocation is used.

  The default SECQTY is 10% of the PRIQTY, or 3 times the page size, whichever is larger. This default quantity is an efficient use of storage allocation. Choosing a SECQTY value that is too small in relation to the PRIQTY value results in track allocation.
- Consider using the PREFORMAT option of the LOAD and REORG utilities. Use this approach when DB2 pre-formatting delays affect the performance or execution-time consistency of applications that do heavy insert processing and the table size can be predicted for a business processing cycle. If you preformat during LOAD or REORG, DB2 does not have to preformat new pages during execution. When the pre-formatted space is used and when DB2 has to extend the table space, normal data set extending and pre-formatting occurs. Consider pre-formatting only if pre-formatting is causing a measurable delay with the insert processing or causing inconsistent elapsed times for insert applications.

What to do next

Quantify the results of pre-formatting in your environment by assessing the performance both before and after using pre-formatting.

Related concepts:

Primary space allocation (DB2 Administration Guide)

Secondary space allocation (DB2 Administration Guide)

Related tasks:

Improving LOAD performance (DB2 Utilities)

Related reference:

CREATE TABLESPACE (DB2 SQL)

CREATE INDEX (DB2 SQL)

LOAD (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

REORG INDEX (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

PRIMARY QUANTITY field (PRIQTY subsystem parameter) (DB2 Installation and Migration)

SECONDARY QTY field (SECQTY subsystem parameter) (DB2 Installation and Migration)

Avoiding excessively small extents

Data set extent size affects performance because excessively small extents can degrade performance during a sequential database scan.

About this task

Suppose that the sequential data transfer speed is 100 MB per second and that the extent size is 10 MB. The sequential scan must move to a new extent ten times per second.

Procedure

To optimize extent sizes, use any of the following approaches:

- Maintain extent sizes that are large enough to avoid excessively frequent extent moving during scans. Because as many as 16 cylinders can be pre-formatted at the same time, keep the extent size greater than 16 cylinders for large data sets.
- Monitor the number of extents to avoid reaching the maximum number of extents on a volume and the maximum number of extents on all volumes. An SMS-managed linear data set is limited to 123 extents on a volume and 7257 total extents on all volumes. A non-SMS-managed data set is limited to 123 extents on a volume and 251 total extents on all volumes. If a data set grows, and extents are not monitored, jobs eventually fail due to these extent limitations.
- Specify sufficient primary and secondary allocations for frequently used data. Doing so minimizes I/O time, because the data is not at different places on the disks.

GUPI

You can take one of the following actions to prevent wasted space for non-partitioned indexes:

- Let DB2 use the default primary quantity and calculate the secondary quantities. By specifying 0 for the IXQTY subsystem parameter. Then omit PRIQTY and SECQTY values in the CREATE INDEX statement or ALTER INDEX statement. If a primary and secondary quantity are specified for an index, you can specify PRIQTY -1 and SECQTY -1 to change to the default primary quantity and calculated secondary quantity.
- If the MGEXTSZ subsystem parameter is set to NO, so that you control secondary space allocations, make sure that the value of PRIQTY + (*N* × SECQTY) is a value that evenly divides into PIECESIZE.

GUPI

- List the catalog or VTOC occasionally to determine the number of secondary allocations for frequently used data sets. Alternatively, you can use IFCID 0258 in the statistics class 3 trace and real-time statistics to monitor data set extensions. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS monitors IFCID 0258.

Related concepts:

How DB2 extends data sets (DB2 Administration Guide)

Related reference:

INDEX SPACE ALLOCATION field (IXQTY subsystem parameter) (DB2 Installation and Migration)

PRIMARY QUANTITY field (PRIQTY subsystem parameter) (DB2 Installation and Migration)

SECONDARY QTY field (SECQTY subsystem parameter) (DB2 Installation and Migration)

OPTIMIZE EXTENT SIZING field (MGEXTSZ subsystem parameter) (DB2 Installation and Migration)

CREATE INDEX (DB2 SQL)

ALTER INDEX (DB2 SQL)

Enabling index I/O parallelism for INSERT operations

You can enable I/O parallelism for INSERT operations to improve insert workload performance on tables with multiple indexes.

About this task

When you insert new data into a table with multiple indexes, each index must be updated individually. However, If you enable index I/O parallelism, the indexes can be updated in parallel. DB2 can use index I/O parallelism only on universal table spaces and partitioned table spaces.

Usually index I/O parallelism is used only for three or more indexes. However, index I/O parallelism can be used for two indexes for tables that are defined with the following options:

- APPEND

- MEMBER CLUSTER
- ORGANIZE BY HASH

Procedure

To enable parallel index I/O:

Specify YES for the value of the INDEX_IO_PARALLELISM subsystem parameter.

What to do next

If I/O parallelism is enabled and IFCID 0358 is turned on, DB2 writes a record to IFCID 0358 when a parallel I/O index insert completes. The record contains the database ID and OBID of the table, and a page set ID of the index that was updated last.

Related concepts:

Hash access on tables (DB2 Administration Guide)

Member affinity clustering (DB2 Data Sharing Planning and Administration)

Parallel processing

Related tasks:

Inserting rows at the end of a partition (DB2 Administration Guide)

Related reference:

INDEX_IO_PARALLELISM in macro DSN6SPRM (DB2 Installation and Migration)

Chapter 5. Configuring storage for performance

Increasing the I/O rate and decreasing the frequency of disk access to move data between real storage and storage devices is key to good performance.

About this task

To meet the diverse needs of application data, a range of storage options is available, each with different access speeds, capacities, and costs per megabyte.

This broad selection of storage alternatives supports requirements for enhanced performance and expanded online storage options, providing more options in terms of performance and price.

The levels in the DB2 storage hierarchy include real storage, storage controller cache, disk, and auxiliary storage.

Related concepts:

What is virtual storage? (z/OS basic skills)

Storage requirements for DB2 (DB2 Installation and Migration)

Related tasks:

Managing I/O processing, response time, and throughput

Minimizing the use of real and virtual storage

You can use several techniques to minimize the use of storage by DB2.

About this task

The amount of real storage often needs to be close to the amount of virtual storage.

Real storage refers to the processor storage where program instructions reside while they are executing. Real storage also refers to where data is held, for example, data in DB2 buffer pools that has not been paged out to auxiliary storage, the EDM pools, and the sort pool. To be used, data must either reside or be brought into processor storage or processor special registers. The maximum amount of real storage that one DB2 subsystem can use is the real storage of the processor, although other limitations might be encountered first.

The large capacity for buffers in real storage and the write avoidance and sequential access techniques allow applications to avoid a substantial amount of read and write I/O, combining single accesses into sequential access, so that the disk devices are used more effectively.

Virtual storage is auxiliary storage space that can be regarded as addressable storage because virtual addresses are mapped to real addresses.

Proper tuning of your buffer pools, EDM pools, RID pools, and sort pools can improve the response time and throughput for your applications and provide optimum resource utilization. Using data compression can also improve buffer-pool hit ratios and reduce table space I/O rates.

Procedure

To minimize the amount of storage that DB2 uses:

- Use less buffer pool storage. Using fewer and smaller buffer pools reduces the amount of real storage space DB2 requires. Buffer pool size can also affect the number of I/O operations performed; the smaller the buffer pool, the more I/O operations needed. Also, some SQL operations, such as joins, can create a result row that does not fit on a 4-KB page.
- Commit frequently to minimize the storage needed for locks.
- Improve the performance for sorting. The highest performance sort is the sort that is avoided. However, because some sorting cannot be avoided, make sorting as efficient as possible. For example, assign the buffer pool for your work file table spaces in database DSNDB07, which are used in sorting, to a buffer pool other than BP0, such as to BP07.
- Provide for pooled threads. Distributed threads that are allowed to be pooled use less storage than inactive database access threads. On a per connection basis, pooled threads use even less storage than inactive database access threads.
- Ensure ECSA size is adequate. The extended common service area (ECSA) is a system area that DB2 shares with other programs. Shortage of ECSA at the system level leads to use of the common service area.

  DB2 places some load modules and data into the common service area. These modules require primary addressability to any address space, including the address space of the application. Some control blocks are obtained from common storage and require global addressability.
- Ensure EDM pool space is being used efficiently. Monitor your use of EDM pool storage using DB2 statistics.
- Use the long-term page fix option for I/O intensive bufferpools. Use PGFIX(YES) for buffer pools with a high I/O rate, that is, a high number of pages read or written.
- Specify an appropriate value for the MAXKEEPD subsystem parameter. A larger value might improve the performance of applications that are bound with the KEEPDYNAMIC(YES) option but also keep SQL statement storage allocated when it is not in use. For systems with real-storage constraints, minimizing the value of the MAXKEEPD might reduce storage use.

Related concepts:

Storage requirements for DB2 (DB2 Installation and Migration)

Storage estimates for data sharing environments (DB2 Installation and Migration)

Read operations and prefetch I/O

Sort access

EDM storage

Common service area storage requirements (DB2 Installation and Migration)

Related tasks:

Monitoring I/O and storage

Tuning database buffer pools

Designing EDM storage space for performance

Managing RID pool size

Improving the performance of sort processing

Fixing a buffer pool in real storage

Calculating EDM pool sizes (DB2 Installation and Migration)

Related reference:

MAX KEPT DYN STMTS field (MAXKEEPD subsystem parameter) (DB2 Installation and Migration)

Storage servers and channel subsystems

Channels can be more of a bottleneck than any other component of the I/O subsystem with IBM TotalStorage and other newer storage servers.

The degree of I/O parallelism that can be sustained efficiently is largely a function of the number of channels. In general, more channels mean better performance.

However, not all channels are alike. ESCON channels, which used to be the predominant channel type, have a maximum instantaneous data transfer rate of approximately 17 MB per second. FICON® channels currently have a speed of 4 GB per second. FICON is the z/OS equivalent of Open Systems Fibre Channel Protocol (FCP). The FICON speed is bidirectional, theoretically allowing 4 GB per second to be sustained in both directions. Channel adaptors in the host processor and the storage server limit the actual speed. The FICON channels in the System z9® and System z10® servers are faster than those in the prior processors, and they feature MIDAW (Modified Indirect Data Address Word) channel program improvements.

Related concepts:

Mainframe hardware: I/O connectivity (z/OS basic skills)

Mainframe hardware: Disk devices (z/OS basic skills)

Balancing the storage controller cache and buffer resources

The amount of cache to use for DB2 depends primarily on the relative importance of price and performance.

Having large memory resources for both DB2 buffers and storage controller cache in not often effective. If you decide to concentrate on the storage controller cache for performance gains, use the maximum available cache size. If the cache is substantially larger than the DB2 buffer pools, DB2 can make effective use of the cache to reduce I/O times for random I/O. For sequential I/O, the improvement that the cache provides is generally small.

Related tasks:

Tuning database buffer pools

Tuning database buffer pools

Buffer pools require monitoring and tuning. Buffer pool sizes are critical to the performance characteristics of an application or group of applications that access data in those buffer pools.

About this task

Introductory concepts

Buffer pools (Introduction to DB2 for z/OS)

The role of buffer pools in caching data (Introduction to DB2 for z/OS)

Buffer pools are areas of virtual storage that temporarily store pages of table spaces or indexes.

When an application program accesses a row of a table, DB2 places the page that contains that row in a buffer. Access to data in this temporary storage is faster than accessing data on a disk. If the required data is already in a buffer, the application program does not need to wait for it to be retrieved from disk, so the time and cost of retrieving the page is reduced.

If the row is changed, the data in the buffer must be written back to disk eventually. But that write operation might be delayed until DB2 takes a checkpoint, or until one of the related write thresholds is reached. (In a data sharing environment, however, the writing mechanism is somewhat different. .) The data remains in the buffer until DB2 decides to use the space for another page. Until that time, the data can be read or changed without a disk I/O operation.

Buffer pools reside in the DBM1 address space. The maximum size of a buffer pool is 16 TB.

Buffer Pool Analyzer: You can use the Buffer Pool Analyzer for z/OS to get recommendations buffer pool allocation changes and to do “what if” analysis of your buffer pools.

Procedure

To change the size and other characteristics of a buffer pool, or enable DB2 automatic buffer pool size management:

Use the ALTER BUFFERPOOL command. You can issue the ALTER BUFFERPOOL command at any time while DB2 is running.

Related concepts:

Making buffer pools large enough for the workload

Tuning group buffer pools (DB2 Data Sharing Planning and Administration)

Read operations and prefetch I/O

Write operations

Introduction to Buffer Pool Analyzer (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related tasks:

Monitoring and tuning buffer pools by using online commands

Managing I/O processing, response time, and throughput

Calculating buffer pool size (DB2 Installation and Migration)

Using Buffer Pool Analyzer (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Facilities and tools for DB2 performance monitoring

DB2 Buffer Pool Analyzer for z/OS

Buffer pool pages

DB2 functions that process data do not directly access page sets on disk. Instead, they access virtual copies of the pages that are held in memory in *buffer pool pages.*

The rows in a table are held inside these pages. Pages are the unit of management within a buffer pool. The unit of I/O is one or more pages chained together.

DB2 makes a *getpage* request operation whenever there is a need to access data in a page, either in an index or a table, such as when executing an SQL statement. DB2 uses random, sequential and list getpage requests. The type of getpage request is determined during the bind process for static SQL and the prepare process for dynamic SQL, and depends on the access path that DB2 chooses to satisfy the SQL request. DB2 checks whether the page requested is in the local buffer pool or is currently being read by a prefetch engine, in which case DB2 waits for the prefetch to complete. If the requested page is not found in the local buffer pool in a data sharing environment, then the global buffer pool is also checked. If the page is still not found, a synchronous I/O is scheduled.

The pages in a database buffer pool can be classified into the following types of pages:

In-use pages
Pages that contain data that is currently being read or updated. This group of pages is important because insufficient space for these pages might cause DB2 to queue or even stop work. These pages are not available to be overwritten, or *stolen*, by a new page of data that is read into the buffer pool.

Updated pages
Pages that contain data that has been updated but not yet written to disk storage. These data on the pages may be reused by the same thread in the same unit of work and by any other thread if row locking is used and the separate threads do not lock the same row. However, these pages are not available to be stolen and overwritten when a new page of data is read into the bufferpool.

Available pages
Pages that contain data that can be considered for reuse to avoid I/O and can be overwritten by data from a new different page that has to be read into the buffer pool. Available pages are normally stolen on a least recently used basis, but you can also specify a first-in-first-out (FIFO) page-stealing algorithm. An important subset of the available pages consists of those that have been prefetched into the pool by a sequential, list, or dynamic prefetch, but have not yet been used. These pages, like other available pages are available for page stealing. When an available page is stolen before it is used and is subsequently needed, DB2 schedules a synchronous I/O operation to read the page into the buffer pool.

Related concepts:

Read operations and prefetch I/O

Write operations

Related tasks:

Choosing a page-stealing algorithm

Deciding how many buffer pools to use

You can assign all objects of each page size to the corresponding default buffer pool. Alternatively, you can specify more buffer pools of each size, according to your specific situation and requirements.

About this task

DB2 creates default buffer pools for each of page size and more default buffer pools for indexes, LOB data, and XML data. The default buffer pools are specified in the subsystem parameters on the DSNTIP1 installation panel.

It is best to separate the buffer pools for DB2 catalog and directory from the buffer pools for user data to isolate catalog and directory activities. In most cases, choose buffer pools other than the default buffer pools for user data, user indexes, and work files.

Procedure

Use one of the following approaches when you create or modify buffer pools:

- Segregate different activities and data into separate buffer pools to achieve better performance and to obtain relatively inexpensive performance diagnosis data from statistics and accounting traces. For example, the following buffer pool assignment strategy is a good starting place:
  - Isolate the catalog and directory from user data or indexes. Objects in the catalog and directory always use the following buffer pools, and the assignments cannot be changed: BP0, BP8K0, BP16K0, and BP32K.
  - Specify separate buffer pools for work files, for 4-KB and 32-KB objects. You can change the assignments by issuing ALTER TABLESPACE statements for the work file table spaces and specifying the assignments in the BUFFERPOOL option.
  - Specify default buffer pools for user data for 4-KB (TBSBPOOL), 8-KB (TBSBP8K) , (TBSB16K), and 32-KB (TBSB32K) objects as needed. Accepting the default values for these parameters means that the same buffer pools are used for user data for and catalog and directory objects.
  - Set the buffer pool names in the IDXBPOOL subsystem parameter for user indexes as needed. The separate buffer pool for indexes might improve random access through an index to data by ensuring all or most of index non-leaf pages are cached in the buffer pool.
  - Set the value of the TBSBPLOB subsystem parameter to non-default value to specify a default buffer pool other than BP0 for LOB table spaces.
  - Set the value of the TBSBPXML subsystem parameter to a non-default value to specify a 16-KB buffer pool other than BP16K0 for XML table spaces.
- For more performance optimization, you might use any of the following approaches to configure more buffer pools. More granular buffer pool assignments can provide better performance and monitoring. However, a balanced approach is best. Too much granularity in your buffer pools can fragment your real storage and increase the cost of managing the system.
  - You can create one or a few in-memory buffer pools to store the frequently accessed data and indexes. If the buffer pools can be large enough to completely cache the objects that are assigned to the pool, consider specifying the PGSTEAL(NONE) option for better performance.
  - You can customize buffer pool parameters to match the characteristics of the data. For example, you might put tables and indexes that are updated

frequently into a buffer pool that has different characteristics from the buffer pools for objects that are infrequently updated. You might keep the objects that are sequentially accessed separate from the objects that are randomly accessed and specify different VPSEQT values for each set of objects.

  - If you encounter large amounts of latch contention that is related to buffer pool pages, you might reduce it by splitting to more buffer pools. Examples of this type of contention include latch class 14 on LRU and hash chains, latch class 23 on adding or removing an entry from deferred write queue, and so forth.
- Choose the default buffer pools for each page size only under the following conditions:
  - Systems that are already constrained by storage
  - When application knowledge that is necessary for more specialized tuning is unavailable.
  - For test systems

Related tasks:

Assigning database objects to buffer pools

Calculating buffer pool size (DB2 Installation and Migration)

Choosing a page-stealing algorithm

Related reference:

DSNTIP1: Buffer pool sizes panel 1 (DB2 Installation and Migration)

Buffer pool thresholds that you can change

Assigning database objects to buffer pools

How you assign data to buffer pools can have a significant impact on performance.

About this task

Objects in the catalog and directory always use the following buffer pools, and the assignments cannot be changed: BP0, BP8K0, BP16K0, and BP32K.

It is best to separate the buffer pools for DB2 catalog and directory from the buffer pools for user data to isolate catalog and directory activities. In most cases, choose buffer pools other than the default buffer pools for user data, user indexes, and work files.

Procedure

To assign database objects to particular buffer pools:

- For table spaces and indexes, issue one of the following SQL statements and specify the BUFFERPOOL option:
  - CREATE TABLESPACE (DB2 SQL)
  - ALTER TABLESPACE (DB2 SQL)
  - CREATE INDEX (DB2 SQL)
  - ALTER INDEX (DB2 SQL)
  - CREATE DATABASE (DB2 SQL)
  - ALTER DATABASE (DB2 SQL)
- For work files, issue an ALTER TABLESACE statement to change the buffer pool assignment of that table spaces. BP0 is the default buffer pool for sorting. It has

a default size of 20000 and a minimum size of 2000. As with any other buffer pool, you can use the ALTER BUFFERPOOL command to change the size of BP0.

Results

The buffer pool is allocated when the table space or index that is assigned to it is first opened.

Related tasks:

Deciding how many buffer pools to use

Calculating buffer pool size (DB2 Installation and Migration)

Choosing buffer pool sizes

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

DSNTIP1: Buffer pool sizes panel 1 (DB2 Installation and Migration)

DSNTIP2: Buffer pool sizes panel 2 (DB2 Installation and Migration)

Buffer pool thresholds

How DB2 uses of a buffer pool is governed by several preset values called thresholds.

Each *buffer pool threshold* is a level of use which, when exceeded, causes DB2 to take some action. Certain thresholds might indicate a buffer pool shortage problem, while other thresholds merely report normal buffer management by DB2. The level of use is usually expressed as a percentage of the total size of the buffer pool. For example, the "immediate write threshold" of a buffer pool is set at 97.5%. When the percentage of unavailable pages in a buffer pool exceeds that value, DB2 writes pages to disk when updates are completed.

For very small buffer pools, of fewer than 1000 buffers, some of the thresholds might be lower to prevent "buffer pool full" conditions, but those thresholds are not described.

Related concepts:

Write operations

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Fixed buffer pool thresholds

Some buffer pool thresholds cannot be changed change. You can monitor buffer pool usage and note how often the fixed thresholds are reached.

If the fixed thresholds are reached too often, the remedy is to use the ALTER BUFFERPOOL command to increase the size of the buffer pool. However, increasing the size of a buffer pool can affect other buffer pools, depending on the total amount of real storage that available for your buffers.

The fixed thresholds are more critical for performance than the variable thresholds. Generally, it is best to set buffer pool sizes large enough to avoid reaching any of the fixed thresholds, except occasionally.

Each of the fixed thresholds is expressed as a percentage of the buffer pool that might be occupied by unavailable pages. From the highest value to the lowest value, DB2 uses the following fixed buffer pool thresholds:

Immediate write threshold: 97.5%

The *immediate write threshold* is checked whenever a page is updated. If the threshold is exceeded, the updated page is written to disk as soon as the update completes. The write is synchronous with the SQL request; that is, the request waits until the write is completed. The two operations do not occur concurrently.

Reaching this threshold has a significant effect on processor usage and I/O resource consumption. For example, updating three rows per page in 10 sequential pages ordinarily requires one or two write operations. However, when the immediate write threshold is exceeded, the updates require 30 synchronous writes.

Sometimes DB2 uses synchronous writes even when the immediate write threshold was not exceeded. For example, when more than two checkpoints pass before a page is written, DB2 uses synchronous writes. Situations such as these do not indicate a buffer shortage.

Data management threshold: 95%

The *data management threshold* is checked before a page is read or updated. If the threshold is not exceeded, DB2 accesses each page in the buffer pool only one time, no matter how many rows are retrieved or updated in the page. If the threshold is exceeded, DB2 accesses the page in the buffer pool one time for each row that is retrieved or updated in that page.

Recommendation: Avoid reaching the data management threshold because it has a significant effect on processor usage.

The data management threshold is maintained for each individual buffer pool. When the data management threshold is reached in one buffer pool, DB2 does not release pages from other buffer pools.

Prefetch threshold: 90%

The *prefetch threshold* is checked at two different times:

- Before a prefetch operation is scheduled. If the prefetch threshold is exceeded, the prefetch is not scheduled.
- During buffer allocation for an already-scheduled prefetch operation, the prefetch is canceled if the prefetch threshold is exceeded.

When the prefetch threshold is reached, prefetch is inhibited until more buffers become available. Operations that use prefetch, such as operations that use large and frequent scans, are adversely affected.

Related concepts:

Write operations

Related tasks:

Monitoring buffer pools (DB2 Administration Guide)

Using OMEGAMON to monitor buffer pool statistics

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Buffer pool thresholds that you can change

You can change some thresholds directly by using the ALTER BUFFERPOOL command.

Introductory concepts

The role of buffer pools in caching data (Introduction to DB2 for z/OS)

GUPI

Changing a threshold in one buffer pool has no effect on any other buffer pool.

You can change the following buffer pool thresholds:

Sequential steal threshold (VPSEQT)

This threshold is a percentage of the buffer pool that might be occupied by sequentially accessed pages. These pages can be in any state: updated, in-use, or available. Hence, any page might or might not count toward exceeding any other buffer pool threshold.

The default value for this threshold is 80%. You can change that to any value from 0% to 100% by using the VPSEQT option of the ALTER BUFFERPOOL command.

This threshold is checked before stealing a buffer for a sequentially accessed page instead of accessing the page in the buffer pool. If the threshold has been exceeded, DB2 tries to steal a buffer that holds a sequentially accessed page rather than one that holds a randomly accessed page.

Setting the threshold to 0% disables prefetch. Any sequentially accessed pages are discarded as soon as the number of available buffers is exceeded by the number of objects being accessed. Setting VPSEQT to 0% is recommended for avoiding unnecessary prefetch scheduling when the pages are already in buffer pool, such as in the case of in-memory indexes or data. However, setting VPSEQT to 0 might disable parallelism.You can achieve the same result without disabling parallelism by using the PGSTEAL NONE option of the ALTER BUFFERPOOL command.

Setting the threshold to 100% allows sequential pages to monopolize the entire buffer pool.

Virtual buffer pool parallel sequential threshold (VPPSEQT)

This threshold is a portion of the buffer pool that might be used to support parallel operations. It is measured as a percentage of the sequential steal threshold (VPSEQT). Setting VPPSEQT to zero disables parallel operation.

The default value for this threshold is 50% of the sequential steal threshold (VPSEQT). You can change that to any value from 0% to 100% by using the VPPSEQT option on the ALTER BUFFERPOOL command.

Virtual buffer pool assisting parallel sequential threshold (VPXPSEQT)

This threshold is a portion of the buffer pool that might be used to assist with parallel operations initiated from another DB2 in the data sharing group. It is measured as a percentage of VPPSEQT. Setting VPXPSEQT to zero disallows this DB2 subsystem from assisting with Sysplex query parallelism at run time for queries that use this buffer pool.

The default value for this threshold is 0% of the parallel sequential threshold (VPPSEQT). You can change that to any value from 0% to 100% by using the VPXPSEQT option on the ALTER BUFFERPOOL command.Sysplex query parallelism is deprecated and is likely to be removed in a future release.

Deferred write threshold (DWQT)

This threshold is a percentage of the buffer pool that might be occupied by unavailable pages, including both updated pages and in-use pages.

The default value for this threshold is 30%. You can change that to any value from 0% to 90% by using the DWQT option on the ALTER BUFFERPOOL command.

DB2 checks this threshold when an update to a page is completed. If the percentage of unavailable pages in the buffer pool exceeds the threshold, write operations are scheduled for enough data sets (at up to 128 pages per data set) to decrease the number of unavailable buffers to 10% below the threshold. For example, if the threshold is 50%, the number of unavailable buffers is reduced to 40%.

When the deferred write threshold is reached, the data sets with the oldest updated pages are written asynchronously. DB2 continues writing pages until the ratio goes below the threshold.

Vertical deferred write threshold (VDWQT)

This threshold is similar to the deferred write threshold, but it applies to the number of updated pages for a single page set in the buffer pool. If the percentage or number of updated pages for the data set exceeds the threshold, writes are scheduled for that data set, up to 128 pages.

You can specify this threshold in one of two ways:

Percentage

Percentage of the buffer pool that might be occupied by updated pages from a single page set. The default value for this threshold is 5%. You can change the percentage to any value from 0% to 90%.

Absolute number

The total number of buffers in the buffer pools that might be occupied by updated pages from a single page set. You can specify the number of buffers from 0 to 9999. If you want to use the number of buffers as your threshold, you must set the percentage threshold to 0.

You can change the percent or number of buffers by using the VDWQT keyword on the ALTER BUFFERPOOL command.

Because any buffers that count toward VDWQT also count toward DWQT, setting the VDWQT percentage higher than DWQT has no effect: DWQT is reached first, write operations are scheduled, and VDWQT is never reached. Therefore, the ALTER BUFFERPOOL command does not allow you to set the VDWQT percentage to a value greater than DWQT. You can specify a number of buffers for VDWQT than is higher than DWQT, but again, with no effect.

GBP dependency causes the threshold to be a constant 64 pages to reduce the number of pages that are written to the group buffer pool at commit.

This threshold is overridden by certain DB2 utilities, which use a constant limit of 64 pages rather than a percentage of the buffer pool size. LOAD, REORG, and RECOVER use a constant limit of 128 pages.

VDWQT set to 0:

If you set VDWQT to zero, DB2 implicitly uses the smaller of 1% of the buffer pool (a specific number of pages), or the number determined by the buffer pool page size as shown in the following table, to avoid synchronous writes to disk.

Table 8. Number of changed pages based on buffer pool size

Buffer pool page size	Number of changed pages
4 KB	40
8 KB	24
16 KB	16
32 KB	12

GUPI

Related concepts:

Buffer pool threshold for parallelism assistants (DB2 Data Sharing Planning and Administration)

Group buffer pool thresholds (DB2 Data Sharing Planning and Administration)

Related tasks:

Choosing a page-stealing algorithm

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Guidelines for setting buffer pool thresholds

How you set buffer pools depends on your workload and the type and size of data being cached. But always think about the entire system when making buffer pool tuning decisions.

GUPI

For additional help in tuning your buffer pools, try the Buffer Pool Analyzer for z/OS.

Frequently re-referenced and updated pages

Suppose that you have a workload such as a branch table in a bank that contains a few hundred rows and is updated by every transaction. For such a workload, you want a high value for the deferred write and vertical deferred write threshold (90%). The result is that I/O is deferred until a checkpoint and you have a lower I/O rate to disk, which helps to keep the hot pages in the buffer pool and avoids the need to write to disk frequently.

However, if the set of pages updated exceeds the size of the buffer pool, setting both DWQT and VDWQT to 90% might cause the sequential prefetch threshold (and possibly the data management threshold and the immediate write threshold) to be reached frequently. You might need to set DWQT and VDWQT lower in that case, or increase the size of the buffer pool.

Rarely referenced pages

Suppose that you have a customer table in a bank that has millions of rows that are accessed randomly or are updated sequentially in batch.

In this case, lowering the DWQT or VDWQT thresholds (perhaps down to 0) can avoid a surge of write I/Os caused by DB2 checkpoint. Lowering those thresholds causes the write I/Os to be distributed more evenly over time. Secondly, this can improve performance for the storage controller cache by avoiding the problem of flooding the device at checkpoints.

Query-only buffer pools

For a buffer pool that is used exclusively for sequential processing, setting VPSEQT to 99% is reasonable and also might enable DB2 to keep space maps in the buffer. If parallel query processing is a large part of the workload, set VPPSEQT and, if applicable, VPXPSEQT, to a very high value. If you are unsure about which values to use for other buffer pool thresholds, use the default values.

Mixed workloads

For a buffer pool used for both query and transaction processing, the value you set for VPSEQT should depend on the respective priority of the two types of processing. The higher you set VPSEQT, the better queries tend to perform, at the expense of transactions. If you are not sure what value to set for VPSEQT, use the default setting.

Buffer pools that contain LOBs

Put LOB data in buffer pools that are not shared with other data. For both LOG YES and LOG NO LOBs, use a deferred write threshold (DWQT) of 0. LOBs specified with LOG NO have their changed pages written at commit time (*force-at-commit* processing). If you set DWQT to 0, those writes happen continuously in the background rather than in a large surge at commit. Dedicating a single buffer pool to LOB objects is especially efficient in data sharing environments.

LOBs defined with LOG YES can use deferred write, but by setting DWQT to 0, you can avoid massive writes at DB2 checkpoints.

Set group buffer pool cast out thresholds to a low value to reduce the need for a large group buffer pools for LOB objects.

PSPI

Related reference:

Buffer pool thresholds that you can change

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

DB2 Buffer Pool Analyzer for z/OS

Choosing buffer pool sizes

The sizes of the buffer pools that your subsystem uses can significantly affect the performance of that subsystem.

About this task

Initially, you set the sizes (in number of pages) of your buffer pools on installation panels DSNTIP1 and DSNTIP2.

However, you can use the ALTER BUFFERPOOL command to modify the sizes of buffer pools. You can also enable automatic buffer pool management.

Procedure

You can use the following approaches to determine the appropriate size for your buffer pools:

- In most cases, specify the largest sizes possible for buffer pools. DB2 handles large buffer pools efficiently. Searching in large buffer pools does not use any more processor resources than searching in smaller pools.

  In general, larger buffer pool sizes provide the following advantages:

  - Result in a higher buffer pool hit ratio, which can reduce the number of I/O operations. Fewer I/O operations can reduce I/O contention, which can provide better response time and reduce the processor resources that are needed for I/O operations.
  - Increase transaction rates with the same response time. For any particular response time, the transaction rate depends greatly on buffer pool size.
  - Prevent I/O contention for the most frequently used disks, particularly the catalog tables, and frequently referenced user tables and indexes. Large buffer pools are beneficial for sort operations. I/O contention on the disks that contain the work file table spaces is reduced.
  - You can use the PGSTEAL(NONE) option with large buffer pools to preload frequently accessed objects into a buffer pool. All pages for the assigned objects remain resident in the buffer pool if they can fit in the allocated space.
- If you see significant paging activity, increase the amount of real storage or decrease the size of the buffer pools. If insufficient real storage exists to back the buffer pool storage, the resulting paging activity might cause performance degradation.

  Important: Insufficient storage causes paging, and in extreme situations, might cause the system to enter wait state and require an IPL of the system.

Related concepts:

Making buffer pools large enough for the workload

The buffer pool hit ratio

Related tasks:

Allocating buffer pool storage to avoid paging

Enabling automatic buffer pool size management

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

DSNTIP1: Buffer pool sizes panel 1 (DB2 Installation and Migration)

DSNTIP2: Buffer pool sizes panel 2 (DB2 Installation and Migration)

Enabling automatic buffer pool size management

You can reduce the amount of time that you spend monitoring and adjusting buffer pools by enabling the DB2 automatic buffer pool size management feature.

About this task

Automatic buffer pool management does not completely replace existing tools to configure, monitor, and tune buffer pool size. However, when you have initially sized your buffer pools, DB2 and WLM can fine tune the buffer pool size, based on long term trends and steady state growth. The DISPLAY BUFFERPOOL output includes an AUTOSIZE attribute. You can enable or disable automatic buffer pool management at the individual buffer pool level. Automatic buffer pool management is off by default.

Procedure

To enable automatic buffer pool size management:

Issue an ALTER BUFFERPOOL command and specify the AUTOSIZE(YES) option. DB2 performs dynamic buffer pool size adjustments that are based on real-time workload monitoring.
When you enable automatic buffer pool management, DB2 reports the buffer pool size and hit ratio for random reads to the z/OS Workload Manager (WLM) component, and automatically increases buffer pool size, as appropriate, by as much as 25% of the originally allocated size.
Whenever the size of a buffer pool is increased, the increased size becomes the new size of the buffer pool. After the buffer pool is deallocated and reallocated, it becomes eligible to be increased by as much as 25% of the new size.

What to do next

Because automatic buffer pool management only increases the size of buffer pools, you might need to sometimes manually reduce the size of a buffer pool that has become too large.

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

MVS Planning: Workload Management (MVS Planning: Workload Management)

Allocating buffer pool storage to avoid paging

DB2 limits the total amount of virtual storage that is allocated for buffer pools to approximately twice the amount of real storage. However, to avoid paging, it is strongly recommended that you set the total buffer pool size to less than the real storage that is available to DB2.

About this task

Paging occurs when the virtual storage requirements for a buffer pool exceed the real storage capacity for the z/OS image. In this case, the least recently used data pages in the buffer pool are migrated to auxiliary storage. Subsequent access to these pages results in a page fault, and the page must be brought into real storage from auxiliary storage. Paging of buffer pool storage can negatively affect DB2 performance. The statistics for PAGE-INS REQUIRED FOR WRITE and PAGE-INS

REQUIRED FOR READ in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report are useful in determining if the buffer pool size setting is too large for available real storage.

If the amount of virtual storage that is allocated to buffer pools is more than twice the amount of real storage, you cannot increase the buffer pool size. DB2 allocates the minimum buffer pool storage for the BP0, BP8K0, BP16K0, and BP32K buffer pools as shown in the following table.

Table 9. Buffer pool storage allocation for BP0, BP8K0, BP16K0, and BP32K

Buffer pool page size	Minimum number of pages allocated
4 KB	2000
8 KB	1000
16 KB	500
32 KB	250

Procedure

To avoid problems with paging:

Set the total buffer pool size to a value that is less than the amount of real storage that is available to DB2.

Related concepts:

What is paging? (z/OS Basic Skills)

What is virtual storage? (z/OS basic skills)

Related reference:

Statistics Report and Trace Blocks (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Choosing a page-stealing algorithm

When DB2 must remove a page from the buffer pool to make room for a newer page, the action is called *stealing* the page from the buffer pool.

About this task

By default, DB2 uses a least-recently-used (LRU) algorithm for managing pages in storage. This algorithm removes pages that have not been recently used and retains recently used pages in the buffer pool. However, DB2 can use different page-stealing algorithms to manage buffer pools more efficiently.

Procedure

To specify the page-stealing algorithm:

1. Determine which page-stealing algorithm is the most efficient for the buffer pool.

Option	Description
PGSTEAL(LRU)	Use this option in most cases. DB2 uses a "least recently used" algorithm to determine when to make buffer pool pages available for stealing. This option keeps pages in the buffer pool that are being used frequently and removes unused pages. It ensures that the most frequently accessed pages are always in the buffer pool. This option has extra costs for LRU chain maintenance and might cause extra latch contention.
PGSTEAL(FIFO)	DB2 uses a "first in first out" algorithm to determine when to make buffer pool pages available for stealing. This option removes the oldest pages in the buffer pool, no matter how frequently they are referenced. This approach to page stealing reduces the cost of determining which pages can be removed, and reduces internal latch contention which result from the LRU algorithm. Specify this option for buffer pools that have no I/O, that is buffer pools that have table space or index entries that always remain in memory.
PGSTEAL(NONE)	DB2 pre-loads the pages into the buffer pool when an object is opened and tries to keep all pages for an object resident in the buffer pool while the object is open. If the buffer pool in not large enough to contain all of the objects, DB2 uses the FIFO algorithm to manage the page stealing.

2. Issue an ALTER BUFFERPOOL command, and specify the PGSTEAL option.

Related concepts:

How z/OS uses physical and virtual storage (z/OS basic skills)

Related tasks:

Monitoring and tuning buffer pools by using online commands

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

Related information:

In-memory table spaces and indexes (DB2 10 for z/OS Performance Topics)

Fixing a buffer pool in real storage

You can use the PGFIX keyword with the ALTER BUFFERPOOL command to fix a buffer pool in real storage for an extended period of time.

About this task

The PGFIX keyword has the following options:

PGFIX(YES)
The buffer pool is fixed in real storage for the long term. Page buffers are fixed when they are first used and remain fixed.

PGFIX(NO)
The buffer pool is fixed in real storage only for the duration of an I/O operation. Page buffers are fixed and unfixed in real storage, allowing for paging to disk. PGFIX(NO) is the default option.

To prevent PGFIX(YES) buffer pools from exceeding the real storage capacity, DB2 uses an 80% threshold when allocating PGFIX(YES) buffer pools. If the threshold is exceeded, DB2 overrides the PGFIX(YES) option with PGFIX(NO).

Procedure

To fix a buffer pool in real storage:

Issue an ALTER BUFFERPOOL command and specify PGFIX(YES). Use PGFIX(YES) for buffer pools that have a high I/O rate, those buffer pools with a high number of pages read or written. For buffer pools with zero I/O, such as some read-only data or some indexes with a nearly 100% hit ratio, PGFIX(YES) is not recommended because it does not provide any performance advantage.

Related concepts:

What is paging? (z/OS Basic Skills)

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

Related information:

DSNB541I (DB2 Messages)

Designing EDM storage space for performance

The environmental descriptor manager (EDM) pools contain skeleton application plans and packages, database descriptors, and cached dynamic SQL statements. You can design them to reduce the number of I/O operations and reduce processing times.

About this task

You can design your EDM storage pools to avoid allocation I/O (a significant part of the total number of I/Os for a transaction), reduce the time that is required to check whether users who attempt to execute a plan are authorized to do so, and reduce the time that is required to prepare statements with the statement cache pool.

When pages are needed from the EDM storage pools, any pages that are available are allocated first. If the available pages do not provide enough space to satisfy the request, pages are “stolen” from an inactive SKCT, SKPT, DBD, or dynamic SQL skeleton. If enough space is still not available, an SQL error code is sent to the application program.

EDM storage pools that are too small cause the following problems.

- Increased I/O activity in DSNDB01.SCT02, DSNDB01.SPT01, DSNDB01.DBD01, DSNDB01.SYSDBDXA, and DSNDB01.SYSSPUXA
- Increased response times, because of loading the SKCTs, SKPTs, and DBDs
- Increased processing and response times time for full prepares of dynamic SQL statements when the EDM statement cache is too small.

Procedure

To ensure the best performance from EDM pools:

Design your EDM storage according to the following table.

Table 10. Designing the EDM storage pools

Design...	To contain...
EDM DBD pool	Database descriptors
EDM statement pool	The cached dynamic SQL statements
EDM skeleton pool	Skeleton copies of plans (SKCTs) and packages (SKPTs)

Related tasks:

➭ Calculating EDM pool sizes (DB2 Installation and Migration)

Related information:

➭ EDM and Dynamic Statement Caching (DB2 for z/OS Best Practices)

EDM storage

The environmental descriptor manager (*EDM*) pools contain skeleton application plans and packages, database descriptors, and cached dynamic SQL statements.

EDM storage is composed of the following components, each of which is in a separate storage area:

EDM DBD pool
An above-the-bar pool that contains database descriptors (DBDs)

EDM statement pool
An above-the-bar pool that contains dynamic cached statements

EDM skeleton pool
An above-the-bar pool that contains skeleton package tables (SKPTs) and skeleton cursor tables (SKCTs)

During the installation process, the DSNTINST CLIST calculates the sizes of the following types of storage:

- EDM statement cache
- EDM DBD cache
- EDM skeleton pool

You can check the calculated sizes on the DSNTIPC installation panel.

For data sharing, you might need to increase the EDM DBD cache storage estimate to compensate for the need to store multiple concurrent DBD copies. Each member maintains a separate copy of a DBD that is referenced by multiple members. New and separate references to the same DBD might result in multiple copies being loaded while invalidated copies remain, until the threads that use them are either committed or deallocated.

Because of an internal process that changes the size of plans initially bound in one release and then are rebound in a later release, you should carefully monitor the sizes of the EDM storage pools, and increase their sizes, if necessary.

Related concepts:

Storage estimate for the EDM pool in a data sharing environment (DB2 Installation and Migration)

Related tasks:

Calculating EDM pool sizes (DB2 Installation and Migration)

Measuring the efficiency of EDM pools

Related reference:

DSNTIPC: CLIST calculations panel 1 (DB2 Installation and Migration)

Measuring the efficiency of EDM pools

You can use information in the DB2 statistics record to calculate the efficiency of the EDM skeleton pool, the EDM DBD cache, and the EDM statement cache.

Procedure

To measure the efficiency of the EDM pools:

Gather the following ratios from the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report:

- DBD HIT RATIO (%)
- CT HIT RATIO (%)
- PT HIT RATIO (%)
- STMT HIT RATIO (%)

These ratios for the EDM pool depend upon your location's workload. In most DB2 subsystems, a value of 80% or more is acceptable. This value means that at least 80% of the requests were satisfied without I/O.
The number of free pages is shown in FREE PAGES. For pools with stealable objects, if this value is more than 20% of the number of pages for the corresponding pools during peak periods, the EDM pool size is probably too large for that type of pool. In this case, you can reduce its size without affecting the efficiency ratios significantly.

EDM pools in the statistics report

The DB2 statistics record provides information on the EDM skeleton pool, the EDM DBD cache, and the EDM statement cache.

The following example shows how Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS presents information about EDM pools in the statistics report.

```
EDM POOL                     QUANTITY
---------------------------  --------
PAGES IN DBD POOL (ABOVE)    14625.00
  HELD BY DBD                  107.00
    STEALABLE PAGES              0.00
  FREE PAGES                 14518.00
```

```
% PAGES IN USE                  0.73
FAILS DUE TO DBD POOL FULL      0.00

PAGES IN STMT POOL (ABOVE)    262.1K
  HELD BY STATEMENTS          207.00
  FREE PAGES                  261.9K
FAILS DUE TO STMT POOL FULL     0.00

PAGES IN SKEL POOL (ABOVE)   2560.00
  HELD BY SKCT                  2.00
  HELD BY SKPT                 12.00
    STEALABLE PAGES            14.00
  FREE PAGES                 2546.00
% PAGES IN USE                  0.00
FAILS DUE TO SKEL POOL FULL     0.00
DBD REQUESTS                  599.3K
DBD NOT FOUND                   0.00
DBD HIT RATIO (%)             100.00
CT REQUESTS                     0.00
CT NOT FOUND                    0.00
CT HIT RATIO (%)                 N/C
PT REQUESTS                  2878.00
PT NOT FOUND                    0.00
PT HIT RATIO (%)              100.00

PKG SEARCH NOT FOUND            0.00
PKG SEARCH NOT FOUND INSERT     0.00
PKG SEARCH NOT FOUND DELETE     0.00

STATEMENTS IN GLOBAL CACHE     48.00
```

Related concepts:

Storage estimate for the EDM pool in a data sharing environment (DB2 Installation and Migration)

Related tasks:

Calculating EDM pool sizes (DB2 Installation and Migration)

Related reference:

DSNTIPC: CLIST calculations panel 1 (DB2 Installation and Migration)

Report Reference (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Calculating the EDM statement cache hit ratio

If you activate the caching of dynamic SQL statements in the dynamic statement cache, the EDM storage statistics provide information that can help you determine how successful your applications are at finding statements in the cache and in the catalog.

About this task

PREPARE REQUESTS (**A**) records the number of requests to search the cache. FULL PREPARES (**B**) records the number of times that a statement was inserted into the cache, which can be interpreted as the number of times a statement was not found in the cache. To determine how often the dynamic statement was used from the cache, check the value in GLOBAL CACHE HIT RATIO (**C**).

Procedure

To calculate the EDM statement cache ratio

Use the following formula:

```
(PREPARE REQUESTS - FULL PREPARES) / PREPARE REQUESTS = hit ratio
```

Example

The following figure shows the dynamic SQL statements part of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report.

```
DYNAMIC SQL,STMT,            QUANTITY
---------------------------  --------
PREPARE REQUESTS  A           8305.3K
  FULL PREPARES   B              0.00
  SHORT PREPARES              8544.5K
GLOBAL CACHE HIT RATIO (%)  C  100.00

IMPLICIT PREPARES                0.00
PREPARES AVOIDED                 0.00
CACHE LIMIT EXCEEDED             0.00
PREP STMT PURGED                 0.00
LOCAL CACHE HIT RATIO (%)         N/C

CSWL - STMTS PARSED              0.00
CSWL - LITS REPLACED             0.00
CSWL - MATCHES FOUND             0.00
CSWL - DUPLS CREATED             0.00
```

Figure 1. EDM storage usage for dynamic SQL statements in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report

For more information, see the IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS for DB2 Performance Expert on z/OS Report Reference.

Related tasks:

Improving dynamic SQL performance by enabling the dynamic statement cache

Related reference:

Statistics Report EDM Pool Activity (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

EDM and Dynamic Statement Caching (DB2 for z/OS Best Practices)

Controlling DBD size for large databases

A database that contains many objects has a larger database descriptor (DBD).

About this task

Introductory concepts

DB2 databases (Introduction to DB2 for z/OS)

If a large number of create, alter, and drop operations are performed on objects in a database with a large DBD, DB2 might encounter more contention from the DBD among transactions that access different objects because storage is not automatically reclaimed in the DBD.

Procedure

To control the size of DBDs for large databases:

- Monitor and manage DBDs to prevent them from becoming too large. Very large DBDs can reduce concurrency and degrade the performance of SQL operations that create or alter objects because of increased I/O and logging. DBDs that are created or altered in DB2 Version 6 or later do not need contiguous storage, but can use pieces of approximately 32 KB. Older DBDs require contiguous storage.
- When you create, alter, and drop objects in a database, use the MODIFY RECOVERY utility to reclaim storage in the DBD. Storage is not automatically reclaimed in a DBD for these operations.

Related concepts:

Objects that are subject to locks

Related tasks:

Reclaiming space in the DBD (DB2 Utilities)

Calculating EDM pool space for database descriptors (DB2 Installation and Migration)

Related reference:

MODIFY RECOVERY (DB2 Utilities)

Managing RID pool size

You can improve the performance of transactions that use the RID pool by specifying a sufficient size for the RID pool.

About this task

DB2 uses the RID pool for all *record identifier (RID) processing*, including the following operations:

- Enforcing unique keys for multi-row updates
- List prefetch, including single index list prefetch access paths
- Multiple index access paths
- Hybrid joins

All concurrent work shares the RID pool. The MAXRBLK subsystem parameter controls the maximum size of the RID pool. The RID pool is created at system initialization, but no space is allocated until RID storage is needed. Space is allocated in 32-KB blocks as needed, until the maximum size that you specify in MAXRBLK is reached.

When RID list processing for any single process requires too much of the space in the RID pool, DB2 might revert to a different access path, such as a table space scan. However, you can also specify that DB2 uses work files to continue RID list processing when the RID pool is not large enough. The maximum size of a single RID list is approximately 26 million RIDs.

DB2 might also revert from list prefetch to table space scans or work file processing at run time if too many rows of a table are accessed for list prefetch to be effective.

Procedure

To manage the size of the RID pool, use any of the following approaches:

- Examine IFCID 0125 in the performance trace to analyze RID pool usage. The RID Pool Processing section of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS record trace report also contains information about RID pool usage.
- Use the following formula to estimate the size needed for the RID pool.

  ```
  Number of concurrent RID processing activities
  ×
    average number of RIDs
  × 2
  × 5 bytes per RID
  ```

  For example, three concurrent transactions that use RID processing, with an average of 4000 RIDs each would require 120 KB of storage:

  ```
  3
  × 4000
  × 2
  × 5 = 120KB
  ```

- If DB2 frequently reverts from access paths that use RID processing to table space scans, take any of the following actions:
  - Increase the maximum size of the RID pool by setting the value of the MAXRBLK subsystem parameter.
  - Set the value of the MAXTEMPS_RID subsystem parameter to enable the use of work files for RID processing. Use the default setting NOLIMIT in most cases. This setting prevents the possibility reverting to table space scans when an arbitrary limit for work file usage is reached.

  Setting the value of the MAXRBLK subsystem parameter to 0 disables all access paths that use RID list processing.

Related concepts:

List prefetch (PREFETCH='L' or 'U')

Hybrid join (METHOD=4)

Multiple index access (ACCESSTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU')

Related reference:

RID POOL SIZE field (MAXRBLK subsystem parameter) (DB2 Installation and Migration)

MAX TEMP RID field (MAXTEMPS_RID subsystem parameter) (DB2 Installation and Migration)

IFCID 125 - RID Pool Processing (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

DB2 for z/OS and List Prefetch Optimizer (IBM Redbooks)

Improving the performance of sort processing

Many factors affect the performance of sort operations, but you can follow certain recommendations to reduce I/O contention and minimize sort row size.

About this task

A sort operation is invoked when a cursor is opened for a SELECT statement that requires sorting. The following factors affect the performance of DB2 sort processing:

- Sort pool size
- I/O contention
- Sort row size
- Whether the data is already sorted

For any SQL statement that initiates sort activity, the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS SQL activity reports provide information on the efficiency of the sort that is involved.

Procedure

To minimize the performance impacts of sort processing, use any of the following approaches:

- Increase the size of the sort pool. The larger the sort pool, the more efficient the sort is. The maximum size of the sort work area allocated for each concurrent sort user depends on the value that you specified for the SRTPOOL subsystem parameter.

  To determine a rough estimate for the maximum sort size, use the following formula:

  ```
  32000
  × (16 + sort key length + sort data length)
  ```

  For sort key length and sort data length, use values that represent the maximum values for the queries you run. To determine these values, refer to the QW0096KL (key length) and QW0096DL (data length) fields in IFCID 0096, as mapped by macro DSNDQW01. You can also determine these values from an SQL activity trace. If a column is in the ORDER BY clause that is not in the select clause, that column should be included in the sort data length and the sort key length as shown in the following example:

  ```
  SELECT C1, C2, C3
   FROM tablex
   ORDER BY C1, C4;
  ```

  If C1, C2, C3, and C4 are each 10 bytes in length, you could estimate the sort pool size as follows:

  ```
  32000
  × (16 + 20 + (10 + 10 + 10 + 10)) = 2342000 bytes
  ```

  The values used in the example above include the items in the following table:

Table 11. Values used in the sort pool size example

Attribute	Value
Maximum number of sort nodes	32000
Size (in bytes) of each node	16
Sort key length (ORDER BY C1, C4)	20
Sort data length (each column is 10 bytes in length)	10+10+10+10

- Minimize I/O contention on the I/O paths to the physical work files, and make sure that physical work files are allocated on different I/O paths and packs to minimize I/O contention. Using disk devices with Parallel Access Volumes

(PAV) support is another way to significantly minimize I/O contention. When I/Os occur in the merge phase of a sort,DB2 uses sequential prefetch to bring pages into the buffer pool with a prefetch quantity of eight pages. However, if the buffer pool is constrained, then DB2 uses a prefetch quantity of four pages or less, or disables prefetch entirely because of the unavailability of enough pages.

- Allocate additional physical work files in excess of the defaults, and put those work files in their own buffer pool.

  Segregating work file activity enables you to better monitor and tune sort performance. It also allows DB2 to handle sorts more efficiently because these buffers are available only for sort without interference from other DB2 work.
- Increase the amount of available space for work files. Applications that use created temporary tables use work file space until a COMMIT or ROLLBACK occurs. (If a cursor is defined WITH HOLD, then the data is held past the COMMIT.) If sort operations happen at the same time that the temporary tables exist, you might need to provide more space for the work files.

  Applications that require star join, materialized views, materialized nested table expressions, non-correlated subqueries or triggers also use work files.
- Write applications to sort only columns that require sorting. Each sorted column in the sort key is counted twice when the sort row size is calculated. A smaller sort row size means that more rows can fit in the sort pool.
- Select VARCHAR columns only when they are required. Varying length columns are padded to their maximum length for sort row size.
- Set the buffer pool sequential steal threshold (VPSEQT) to 99% unless sparse index is used to access the work files. The default value, which is 80%, allows 20% of the buffers to go unused. A value of 99% prevents space map pages, which are randomly accessed, from being overwritten by massive prefetch.
- Increase the buffer pool deferred write threshold (DWQT) or data set deferred write threshold (VDWQT) values. If DWQT or VDWQT are reached, writes are scheduled. For a large sort that uses many logical work files, scheduled writes are difficult to avoid, even if a very large buffer pool is specified. As you increase the value of VDWQT, watch for buffer shortage conditions and either increase the work file buffer pool size or reduce VDWQT if buffer shortages occur.

Related tasks:

Creating additional work file table spaces to reduce contention

Calculating sort pool storage in local storage (DB2 Installation and Migration)

Related reference:

DSNTIPC: CLIST calculations panel 1 (DB2 Installation and Migration)

SORT POOL SIZE field (SRTPOOL subsystem parameter) (DB2 Installation and Migration)

Buffer pool thresholds that you can change

How sort work files are allocated

The work files that are used in sort are *logical work files*, which reside in work file table spaces in your work file database (which is DSNDB07 in a non data-sharing environment).

The sort begins with the input phase, when ordered sets of rows are written to work files. At the end of the input phase, when all the rows have been sorted and inserted into the work files, the work files are merged together, if necessary, into a single work file that contains the sorted data. The merge phase is skipped if only one work file exists at the end of the input phase. In some cases, intermediate merging might be needed if the maximum number of sort work files has been allocated.

DB2 uses the buffer pool when writing to the logical work file. Only the buffer pool size limits the number of work files that can be used for sorting.

A sort can complete in the buffer pool without I/Os operations. This ideal situation might be unlikely, especially if the amount of data being sorted is large. The sort row size is actually made up of the columns being sorted (the length of the sort key) and the columns that the user selects (the length of the sort data). A large buffer pool for sort activity can help you to avoid disk I/O operations.

When your application needs to sort data, DB2 tries to allocate each sort work file on a table space that has the following attributes:

- Is a segmented (non-universal) table space.
- Has no secondary allocation (SECQTY = 0), or is a user-managed table space, regardless of secondary space allocation.

When table spaces that have the preferred attributes are not available, the action taken depends on the value of the WFDBSEP subsystem parameter. If the value is 'YES', the sort operation fails. If the value is 'NO', another available table space is selected.

After the selection based on the table space attributes, DB2 allocates the work files based on the overall record length. When the record length (*data + key + prefix*) is greater than 100 bytes, DB2 attempts to create the work file in a table space with 32 KB page size. If the record length is 100 bytes or less, DB2 prefers a table space with the 4 KB page size.

Finally, the least recently used table space that has the preferred attributes is selected.

Related concepts:

Work file database (Introduction to DB2 for z/OS)

Segmented (non-UTS) table spaces (deprecated) (Introduction to DB2 for z/OS)

Virtual storage requirements for storage pools and working storage (DB2 Installation and Migration)

Secondary space allocation (DB2 Administration Guide)

Related tasks:

Creating additional work file table spaces to reduce contention

Related reference:

Statistics Report Buffer Pool Sort/Merge (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

SECONDARY QTY field (SECQTY subsystem parameter) (DB2 Installation and Migration)

SEPARATE WORK FILES field (WFDBSEP subsystem parameter) (DB2 Installation and Migration)

Related information:

Work file table spaces DB2 10 for z/OS Performance Topics)

Work file sizing (DB2 9 for z/OS Performance Topics)

WORKFILE database enhancements (DB2 9 for z/OS Performance Topics)

Managing the opening and closing of data sets

Having the needed data sets open and available for use is important for the performance of transactions. However, the number of open data sets affects the amount of available storage, and the number of open data sets in read/write state affects restart time.

About this task

DB2 uses a *deferred close* process to delay the physical closing of page sets or partitions until necessary, to avoid extra I/O processing. *Page set* refers to the set of data pages for a table space or index.

Deferred close enables other applications or users to access unused table spaces and associated indexes, without reopening the data sets or partitions.

DB2 dynamically manages page sets by using two levels of closure:

Logical closure
When the application is deallocated from that page set. Logical closure happens at commit or deallocation time, depending on the value of the RELEASE bind option and the use count. When a page set is logically closed, the page set use count is decremented. When the page set use count is zero, the page set is considered not in use, and the page set becomes a candidate for physical closure.

Physical closure
When DB2 closes and deallocates the data sets for the page set.

DB2 defers the closing and de-allocating of open table spaces or indexes until the number of open data sets approaches the value of the DSMAX subsystem parameter. The CLOSE option of the CREATE TABLESPACE and CREATE INDEX statements specifies the priority in which data sets are closed.

Procedure

To control the maximum number of open data sets, use the following approaches:

- Specify an appropriate value for the DSMAX subsystem parameter.
  - Leave enough margin in your specification of DSMAX so that frequently used data sets can remain open after they are no longer referenced. If data sets are opened and closed frequently, such as every few seconds, you can improve performance by increasing DSMAX.
  - Specify values for the PCLOSEN and PCLOSET subsystem parameters to control how long data sets stay open in a read/write state. The number of open data sets on your subsystem that are in read/write state affects checkpoint costs and log volumes.
  - Consider creating segmented table spaces to reduce the number of data sets. This approach is most useful for development or end-user systems that include many smaller tables that can be combined into single table spaces.

When the number of open data sets approaches the value of the DSMAX subsystem parameter, DB2 begins closing page sets. First, page sets or objects that are defined with the CLOSE YES option are closed. The least recently used page sets are closed first.

When more data sets must be closed, DB2 next closes page sets or partitions for objects that are defined with the CLOSE NO option. The least recently used CLOSE NO data sets are closed first.

- Specify the CLOSE NO option for page sets that contain data that must be accessed without the delay of opening the data sets but is accessed only infrequently. For table spaces that are accessed continually, the value of the CLOSE option is unimportant because the data sets remain open. The same is also true, although less so, for table spaces whose data is not referenced for short periods of time. Because DB2 uses deferred close to manage data sets, the data sets are likely to be open when they are used again.
- If the number of open data sets is a concern, choose CLOSE YES for page sets with many partitions or data sets.

Related tasks:

Choosing a RELEASE option

Related reference:

DSMAX field (DSMAX subsystem parameter) (DB2 Installation and Migration)

RO SWITCH CHKPTS field (PCLOSEN subsystem parameter) (DB2 Installation and Migration)

RO SWITCH TIME field (PCLOSET subsystem parameter) (DB2 Installation and Migration)

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

CREATE INDEX (DB2 SQL)

ALTER INDEX (DB2 SQL)

Related information:

00C20113 (DB2 Codes)

How DB2 determines the initial value of DSMAX

DB2 uses a formula to calculate the initial value for the DSMAX subsystem parameter.

DB2 calculates the initial value of the DSMAX subsystem parameter according to the following formula:

```
databases
× ((tables
×indexes) + table-spaces)
```

databases
The number of concurrent databases specified in the DATABASES field on installation panel DSNTIPE.

tables The number of tables per database specified in the TABLES field on installation panel DSNTIPD.

indexes
The number of indexes per table. The installation CLIST sets this variable to 2.

table-spaces
The number of table spaces per database specified in the TABLE SPACES field on installation panel DSNTIPD.

The calculated value is shown on CLIST calculations panel 1: DSNTIPC

Related reference:

DSMAX field (DSMAX subsystem parameter) (DB2 Installation and Migration)

DATABASES field for panel DSNTIPD (DB2 Installation and Migration)

TABLES field (DB2 Installation and Migration)

TABLE SPACES field (DB2 Installation and Migration)

DSNTIPC: CLIST calculations panel 1 (DB2 Installation and Migration)

Evaluating the value of DSMAX

You might need to increase the value of the DSMAX subsystem parameter to prevent unneeded I/O operations.

Procedure

To evaluate whether to modify the value of the DSMAX subsystem parameter, use the following approaches:

- Specify a value for the DSMAX subsystem parameter that is larger than the number of data sets that are open and in use at one time.
  - For the most accurate count of open data sets, refer to the OPEN/CLOSE ACTIVITY section of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report. Make sure the statistics trace was run at a peak period, so that you can obtain the most accurate maximum figure.
  - The best indicator of when to increase the value of the DSMAX subsystem parameter is when the open and close activity of data sets is high. One-per-second is a general guideline. Refer to the OPEN/CLOSE value under the SER.TASK SWITCH section of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report or NUMBER OF DATASET OPENS in the bufferpool statistics (which provides the statistics for specific buffer pools). Consider increasing DSMAX when these values show more than one event per second.
- Consider partitioned and LOB tables spaces. The formula that DB2 uses to calculate the initial size of the DSMAX subsystem parameter does not account for partitioned or LOB table spaces. Those table spaces can have many data sets.
- Consider the data sets for non-partitioned indexes that are defined on partitioned table spaces with many partitions and multiple partitioned indexes. When those indexes are defined with small PIECESIZE values, many data sets might be the result.
- You can calculate the total number of data sets (rather than the number that are open during peak periods).
  1. To find the number of simple and segmented table spaces, issue the following query:

```
SELECT CLOSERULE, COUNT(*)
  FROM SYSIBM.SYSTABLESPACE
  WHERE PARTITIONS = 0
  GROUP BY CLOSERULE;
```

The calculation assumes that you have one data set for each simple, segmented, and LOB table space. These catalog queries are included in DSNTESP in SDSNSAMP. You can use them as input to SPUFI.

2. To find the number of data sets for the partitioned table spaces, issue the following query:

   ```
   SELECT CLOSERULE, COUNT(*), SUM(PARTITIONS)
     FROM SYSIBM.SYSTABLESPACE
     WHERE PARTITIONS > 0
     GROUP BY CLOSERULE;
   ```

   The query returns the number of partitioned table spaces and the total number of partitions.
3. To find the number of data sets required for each nonpartitioned index, issue the following query:

   ```
   SELECT CLOSERULE, COUNT(*)
     FROM   SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
     WHERE  T1.NAME = T2.IXNAME
     AND    T1.CREATOR = T2.IXCREATOR
     AND    T2.PARTITION = 0
     GROUP BY CLOSERULE;
   ```

   The calculation assumes that you have only one data set for each non-partitioned index. If you use pieces, adjust accordingly.
4. To find the number of data sets for the partitioned indexes, issue the following query:

   ```
   SELECT CLOSERULE, COUNT(*)
     FROM   SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
     WHERE  T1.NAME = T2.IXNAME
     AND    T1.CREATOR = T2.IXCREATOR
     AND    T2.PARTITION > 0
     GROUP BY CLOSERULE;
   ```

   The query returns the number of index partitions. You have one data set for each index partition.
5. To find the total number of data sets, add the numbers that result from the four queries. (For Query 2, use the sum of the partitions that was obtained.)

Related concepts:

How DB2 determines the initial value of DSMAX

Related tasks:

Switching to read-only for infrequently updated and infrequently accessed page sets

Related reference:

DSMAX field (DSMAX subsystem parameter) (DB2 Installation and Migration)

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

RO SWITCH CHKPTS field (PCLOSEN subsystem parameter) (DB2 Installation and Migration)

RO SWITCH TIME field (PCLOSET subsystem parameter) (DB2 Installation and Migration)

Switching to read-only for infrequently updated and infrequently accessed page sets

By converting infrequently used page sets from read-write to read-only, you can improve performance and data recovery by minimizing the amount of logging activities and reducing the number of log records.

About this task

For both CLOSE YES and CLOSE NO page sets, SYSLGRNX entries are updated when the page set is converted from read-write state to read-only state. When this conversion occurs for table spaces, the SYSLGRNX entry is closed and any updated pages are externalized to disk. For indexes defined as COPY NO, no SYSLGRNX entries occur, but the updated pages are externalized to disk.

By converting infrequently used page sets from read-write to read-only state, you can achieve the following performance benefits:

- Improved data recovery performance because SYSLGRNX entries are more precise, closer to the last update transaction commit point. As a result, the RECOVER utility has fewer log records to process.
- Minimized logging activities. Log records for page set open, checkpoint, and close operations are only written for updated page sets or partitions. Log records are not written for read-only page sets or partitions.

Procedure

To specify when unused pages are converted to read only:

Specify the values for the RO SWITCH CHKPTS and RO SWITCH TIME fields of the DSNTIPL installation panel.

RO SWITCH CHKPTS (PCLOSEN subsystem parameter)
The number of consecutive DB2 checkpoints since a page set or partition was last updated.

RO SWITCH TIME (PCLOSET subsystem parameter)
the amount of elapsed time since a page set or partition was last updated.

Use the following recommendations to determine how to set these values:

- In most cases, the default values are adequate. However, if you find that the amount of R/O switching is causing a performance problem for the updates to SYSLGRNX, consider increasing the value of RO SWITCH TIME.
- For table spaces that are defined with the NOT LOGGED option, the values for RO SWITCH CHKPTS and RO SWITCH TIME are set to the recommended value of 1. Changing these values is not recommended. All read-write table spaces that are defined with the NOT LOGGED option and not in use are converted to read-only whenever a DB2 checkpoint occurs. If a checkpoint does not occur, the not logged table spaces are converted to read-only one minute after the commit of the last update. DB2 writes the table space from the buffer pool to external media when it converts the table space from read-write to read-only, externalizing any unprotected modifications to the data.

Related concepts:

➡ The NOT LOGGED attribute (DB2 Administration Guide)

Related tasks:

Choosing a checkpoint frequency

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

RO SWITCH CHKPTS field (PCLOSEN subsystem parameter) (DB2 Installation and Migration)

RO SWITCH TIME field (PCLOSET subsystem parameter) (DB2 Installation and Migration)

SYSIBM.SYSLGRNX table (DB2 SQL)

Improving disk storage

You can configure your storage devices and disk space to ensure better performance from DB2.

Related concepts:

Tips for archiving to disk (DB2 Administration Guide)

Related tasks:

Estimating disk storage for user data (DB2 Administration Guide)

Compressing your data

Designing indexes for performance

Selecting storage devices

Some storage device types are more optimal for certain types of applications.

Procedure

To choose storage devices types:

Consider the following hardware characteristics that affect performance.

- The size of the cache
- The number of channels and type of channels that are attached and online to a group of logical volumes, including high performance FICON
- The size of non-volatile storage (NVS), if deferred write performance is a problem
- Disk arrays
- Advanced features such as Parallel Access Volumes (PAV), HyperPAV, Multiple Allegiance, and FlashCopy®
- Fast remote replication techniques

Related concepts:

Storage servers and advanced features

Storage servers

An I/O subsystem typically consists of many storage disks, which are housed in *storage servers* such as the IBM TotalStorage DS8000®.

Storage servers provide increased functionality and performance over that of “Just a Bunch of Disks” technology.

Cache is one of the additional functions. Cache acts as a secondary buffer as data is moved between real storage and disk. Storing the same data in processor storage and the cache is not useful. To be useful, the cache must be significantly larger than the buffers in real storage, store different data, or provide another performance advantage. TotalStorage and many other new storage servers use large caches and always pre-stage the data in the cache. You do not need to actively manage the cache in the newer storage servers as you must do with older storage device types.

With IBM TotalStorage and other new storage servers, disk performance does not generally affect sequential I/O performance. The measure of disk speed in terms of RPM (revolutions per minute) is relevant only if the cache hit ratio is low and the I/O rate is very high. If the I/O rate per disk is proportional to the disk size, small disks perform better than large disks. Large disks are very efficient for storing infrequently accessed data. As with cache, spreading the data across more disks is always better.

Remote replication is a significant factor in stored performance. When I/O performance problems occur, especially in when remote replication is involved, investigate the storage systems and communications network before looking for problems with the host database system.

Storage servers and advanced features

IBM TotalStorage offers many advanced features to further boost performance.

Extended address volumes (EAV)
With extended address volumes (EAV), you can store more data that is in VSAM data sets on a single volume than you can store on non-extended address volumes. The maximum amount of data that you can store in a single DB2 table space or index space is the same for extended and non-extended address volumes. The same DB2 data sets might use more space on extended address volumes than on non-extended address volumes because space allocations in the extended area are multiples of 21 cylinders on extended address volumes.

Parallel Access Volumes (PAV)
Enables multiple concurrent I/O operations on a device when the I/O requests originate from the same system. Parallel access volumes (PAV) make storing multiple partitions on the same volume with almost no loss of performance possible. In older disk subsystems, if more than one partition is placed on the same volume (intentionally or otherwise), attempts to read the partitions result in contention. The contention shows up as I/O subsystem queue time. Without PAVs, poor placement of a single data set can almost double the elapsed time of a parallel query.

Multiple allegiance
Enables multiple active concurrent I/O operations on a single device when the I/O requests originate from different systems. Together, parallel access volumes (PAVs) and multiple allegiance dramatically improve I/O performance for parallel work on the same volume. These features nearly eliminate I/O subsystem queue or PEND time and lower elapsed time for transactions and queries.

FlashCopy
Provides for fast copying of full volumes. After an initialization period is complete, the logical copy is considered complete but the physical movement of the data is deferred.

Peer-to-Peer Remote Copy (PPRC)
Provide a faster method for recovering DB2 subsystems at a remote site in the event of a disaster at a local site.

Other storage servers might offer similar functions.

Related tasks:

Backing up with RVA storage control or Enterprise Storage Server (DB2 Administration Guide)

Using disk space effectively

How you allocate and manage data sets, compress your data, and design your indexes can affects the performance of DB2.

Procedure

To use disk space more efficiently, use any of the following approaches:

- Change your allocation of data sets to keep data sets within primary allocations.
- Manage them with the Hierarchical Storage Management functional component (DFSMShsm) of DFSMS.
- Compress your data.
- Choose a page size that gives you good disk use and I/O performance characteristics.
- Evaluate the need for and characteristics of your indexes.

What to do next

To manage the use of disk, you can use RMF™ to monitor how your devices are used. Watch for usage rates that are higher than 30% to 35%, and for disk devices with high activity rates. Log devices can have more than 50% utilization without performance problems.

Related concepts:

DB2 and DFSMS (Introduction to DB2 for z/OS)

Related tasks:

Reserving free space for table spaces

Reserving free spaces for indexes

Compressing your data

Designing indexes for performance

Chapter 22, “Improving performance for LOB data,” on page 273

Choosing data page sizes

Related reference:

z/OS RMF User's Guide

Related information:

Managing DB2 data sets with DFSMShsm (DB2 Administration Guide)

Allocating and extending data sets

Primary and secondary allocation sizes are the main factors that affect the amount of disk space that DB2 uses.

In general, the primary allocation must be large enough to handle the storage needs that you anticipate. The secondary allocation must be large enough for your applications to continue operating until the data set is reorganized.

If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space.

IFCID 0258 allows you to monitor data set extension activities by providing information, such as the primary allocation quantity, maximum data set size, high allocated space before and after extension activity, number of extents before and after the extend, maximum volumes of a VSAM data set, and number of volumes before and after the extend. Access IFCID 0258 in Statistics Class 3 (SC03) through an IFI READA request.

Related concepts:

DB2 space allocation (DB2 Administration Guide)

Related tasks:

Requesting data asynchronously from a monitor program

Related information:

Implementing DB2 storage groups (DB2 Administration Guide)

Planning the placement of DB2 data sets:

To improve performance, plan the placement of DB2 data sets carefully.

Concentrate mainly on data sets for system files (especially the active logs), for the DB2 catalog and directory, and for user data and indexes. The objective is to balance I/O activity between different volumes, control units, and channels. Doing so minimizes the I/O elapsed time and I/O queuing.

Related tasks:

Monitoring work file data sets

Improving DB2 log performance

Managing I/O processing, response time, and throughput

Estimating concurrent I/O requests:

The number of concurrent I/O requests is important when you calculate the number of data paths for your DB2 subsystem.

About this task

DB2 has a multi-tasking structure in which each user's request runs under a different task control block (TCB). In addition, the DB2 system itself has its own TCBs and SRBs for logging and database writes.

Procedure

To estimate the maximum number of concurrent I/O requests when your system is loaded with data:

Use the following formula: MAX USERS + 600 prefetches + 600 asynchronous writes (A maximum of 300 deferred write engines and 300 castout engines asynchronous writes are supported)

Related tasks:

Controlling the number of I/O operations

Related reference:

MAX USERS field (CTHREAD subsystem parameter) (DB2 Installation and Migration)

Identifying crucial DB2 data sets:

When you are placing your data sets, you need to first consider data sets that are crucial for DB2 to function properly.

Procedure

To identify the crucial DB2 data sets:

Use the I/O reports from the DB2 performance trace. If these reports are not available, consider the following data sets to be most important:

For transactions

- DSNDB01.SCT02 and its index
- DSNDB01.SPT01 and its index
- DSNDB01.SYSSPUXA and its index
- DSNDB01.SYSSPUXB and its index
- DSNDB01.DBD01 and its index
- DSNDB01.SYSDBDXA and its index
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH table
- DSNDB06.SYSPKAGE
- Active and archive logs
- Most frequently used user table spaces and indexes

For queries

- DSNDB01.DBD01 and its index
- DSNDB01.SYSDBDXA and its index
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH
- DSNDB06.SYSPKAGE
- DSNDB06.SYSDBASE table space and its indexes
- DSNDB06.SYSVIEWS table space and the index on SYSVTREE
- Work file table spaces
- QMF system table data sets
- Most frequently used user table spaces and indexes

These lists do not include other data sets that are less crucial to DB2 performance, such as those that contain program libraries, control blocks, and formats. Those types of data sets have their own design recommendations.

Related concepts:

DB2 directory (Introduction to DB2 for z/OS)

Reorganizing the catalog (DB2 SQL)

Reorganizing the catalog and directory (DB2 Utilities)

Related reference:

DB2 catalog tables (DB2 SQL)

Changing catalog and directory size and location:

You can change the size or location of your DB2 catalog or directory .

Procedure

To change the size or location of DB2 catalog or directory data sets:

Choose one of the following actions:
- Run the RECOVER utility on the appropriate database
- Run the REORG utility on the appropriate table space

Related concepts:

Reorganizing the catalog (DB2 SQL)

Reorganizing the catalog and directory (DB2 Utilities)

Related reference:

Facilities and tools for DB2 performance monitoring

RECOVER (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

Improving space allocation and pre-formatting

You can improve the performance of applications that use heavy insert processing by controlling how space is allocated and pre-formatted.

About this task

When inserting records, DB2 pre-formats space within a page set as needed. The allocation amount, which is either by *cylinder* or *track*, determines the amount of space that is pre-formatted at any one time.

Because less space is pre-formatted at one time for the track allocation amount, a mass insert can take longer when the allocation amount is track than the same insert when the allocation amount is cylinder. However, smart secondary space allocation minimizes the difference between track and cylinder allocation.

Cylinder allocation can reduce the time required to do SQL mass inserts and to perform LOGONLY recovery. It does not affect the time required to recover a table space from an image copy or to run the REBUILD utility.

Procedure

Use the following approaches to control space allocation and pre-formatting:
- Specify your space allocation amounts to ensure allocation by cylinder. The allocation amount depends on device type and the values that you specify for PRIQTY and SECQTY when you define table spaces and indexes. If you use record allocation for more than a cylinder, cylinder allocation is used.

  The default SECQTY is 10% of the PRIQTY, or 3 times the page size, whichever is larger. This default quantity is an efficient use of storage allocation. Choosing a SECQTY value that is too small in relation to the PRIQTY value results in track allocation.
- Consider using the PREFORMAT option of the LOAD and REORG utilities. Use this approach when DB2 pre-formatting delays affect the performance or execution-time consistency of applications that do heavy insert processing and the table size can be predicted for a business processing cycle. If you preformat during LOAD or REORG, DB2 does not have to preformat new pages during execution. When the pre-formatted space is used and when DB2 has to extend the table space, normal data set extending and pre-formatting occurs. Consider

pre-formatting only if pre-formatting is causing a measurable delay with the insert processing or causing inconsistent elapsed times for insert applications.

What to do next

Quantify the results of pre-formatting in your environment by assessing the performance both before and after using pre-formatting.

Related concepts:

Primary space allocation (DB2 Administration Guide)

Secondary space allocation (DB2 Administration Guide)

Related tasks:

Improving LOAD performance (DB2 Utilities)

Related reference:

CREATE TABLESPACE (DB2 SQL)

CREATE INDEX (DB2 SQL)

LOAD (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

REORG INDEX (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

PRIMARY QUANTITY field (PRIQTY subsystem parameter) (DB2 Installation and Migration)

SECONDARY QTY field (SECQTY subsystem parameter) (DB2 Installation and Migration)

Avoiding excessively small extents

Data set extent size affects performance because excessively small extents can degrade performance during a sequential database scan.

About this task

Suppose that the sequential data transfer speed is 100 MB per second and that the extent size is 10 MB. The sequential scan must move to a new extent ten times per second.

Procedure

To optimize extent sizes, use any of the following approaches:

- Maintain extent sizes that are large enough to avoid excessively frequent extent moving during scans. Because as many as 16 cylinders can be pre-formatted at the same time, keep the extent size greater than 16 cylinders for large data sets.
- Monitor the number of extents to avoid reaching the maximum number of extents on a volume and the maximum number of extents on all volumes. An SMS-managed linear data set is limited to 123 extents on a volume and 7257 total extents on all volumes. A non-SMS-managed data set is limited to 123 extents on a volume and 251 total extents on all volumes. If a data set grows, and extents are not monitored, jobs eventually fail due to these extent limitations.

- Specify sufficient primary and secondary allocations for frequently used data. Doing so minimizes I/O time, because the data is not at different places on the disks.

  GUPI

  You can take one of the following actions to prevent wasted space for non-partitioned indexes:

  – Let DB2 use the default primary quantity and calculate the secondary quantities. By specifying 0 for the IXQTY subsystem parameter. Then omit PRIQTY and SECQTY values in the CREATE INDEX statement or ALTER INDEX statement. If a primary and secondary quantity are specified for an index, you can specify PRIQTY -1 and SECQTY -1 to change to the default primary quantity and calculated secondary quantity.

  – If the MGEXTSZ subsystem parameter is set to NO, so that you control secondary space allocations, make sure that the value of PRIQTY + (*N* × SECQTY) is a value that evenly divides into PIECESIZE.

  GUPI

- List the catalog or VTOC occasionally to determine the number of secondary allocations for frequently used data sets. Alternatively, you can use IFCID 0258 in the statistics class 3 trace and real-time statistics to monitor data set extensions. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS monitors IFCID 0258.

Related concepts:

How DB2 extends data sets (DB2 Administration Guide)

Related reference:

INDEX SPACE ALLOCATION field (IXQTY subsystem parameter) (DB2 Installation and Migration)

PRIMARY QUANTITY field (PRIQTY subsystem parameter) (DB2 Installation and Migration)

SECONDARY QTY field (SECQTY subsystem parameter) (DB2 Installation and Migration)

OPTIMIZE EXTENT SIZING field (MGEXTSZ subsystem parameter) (DB2 Installation and Migration)

CREATE INDEX (DB2 SQL)

ALTER INDEX (DB2 SQL)

Chapter 6. Configuring subsystems for concurrency

You can use system settings and DB2 subsystem parameters to improve concurrency.

Related tasks:

Improving concurrency

Estimating the storage needed for locks

You can estimate the amount of storage needed for locks.

About this task

An estimate of the amount of storage needed for locks is calculated when DB2 is installed. The CLIST stores the greater of 2GB or the calculated value of the NUMLKUS subsystem parameter.

Procedure

To estimate the storage that is required for DB2 locks:

1. Gather the following information:
   - The maximum number of row or page locks per user, in the value of the NUMLKUS subsystem parameter.
   - The maximum number of allied threads that can be active concurrently, in the value of the CTHREAD subsystem parameter.
   - The maximum number of database access threads that can be active concurrently, in the value of the MAXDBAT subsystem parameter.
2. Use the following formula, which assumes that each lock needs 540 bytes of storage:

   ```
   Bytes of Storage = 100MB +
    (CTHREAD + MAXDBAT)

   × NUMLKUS

   × 540 bytes)
   ```

   The result is a high-end estimate of the storage space that is needed for locks, because the formula assumes that the maximum number of users are connected, and each user holds the maximum number of locks.

Related reference:

MAX STORAGE FOR LOCKS field (DB2 Installation and Migration)

MAX USERS field (CTHREAD subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

LOCKS PER USER field (NUMLKUS subsystem parameter) (DB2 Installation and Migration)

Related information:

DSNT438I (DB2 Messages)

IRLM startup procedure options

You can control how DB2 uses locks by specifying certain options when you start the internal resource lock manager (IRLM).

About this task

PSPI

When you issue the z/OS START **irlmproc** command, the values of the options are passed to the startup procedure for the DB2 IRLM. (If an option is not explicitly specified on the command, the value of its corresponding installation parameter is used.)

The options that are relevant to DB2 locking are:

SCOPE
Whether IRLM is used for data sharing (GLOBAL) or not (LOCAL). Use LOCAL unless you are using data sharing. If you use data sharing, specify GLOBAL.

DEADLOK
The two values of this option specify:

1. The number of seconds between two successive scans for a local deadlock
2. The number of local scans that occur before a scan for global deadlock starts

PC
Ignored by IRLM. However, PC is positional and must be maintained in the IRLM for compatibility.

MAXCSA
Ignored by IRLM. However, MAXCSA is positional and must be maintained in the IRLM for compatibility.

The maximum amount of storage available for IRLM locks is limited to 90% of the total space given to the IRLM private address space during the startup procedure. The other 10% is reserved for IRLM system services, z/OS system services, and “must complete” processes to prevent the IRLM address space from abending, which would bring down your DB2 system. When the storage limit is reached, lock requests are rejected with an out-of-storage reason code.

You can use the F irlmproc,STATUS,STOR command to monitor the amount of storage that is available for locks and the MODIFY irlmproc,SET command to dynamically change the maximum amount of IRLM private storage to use for locks.

PSPI

Setting installation options for wait times

These options determine how long it takes DB2 to identify that a process must be timed out or is deadlocked. They affect locking in your entire DB2 subsystem.

Specifying the interval for detecting deadlocks

You can specify the interval at which DB2 scans for deadlocked processes at regular intervals.

About this task

Deadlock detection can cause latch suspensions.

Procedure

To specify the interval for detecting deadlocks:

Specify a value in seconds for the DEADLOCK TIME field on installation panel DSNTIPJ.

- For systems in which deadlocking is not a problem, have deadlock detection run less frequently for the best performance and concurrency (but do not choose a value greater than 5 seconds).
- If your system is prone to deadlocks, you want those detected as quickly as possible. In that case, choose 1.

The default value of the DEADLOCK TIME field is 1 second.

Specifying the amount of inactive time before a timeout

You can specify how long your system waits for suspended processes.

Procedure

To specify the minimum number of seconds before a timeout can occur:

Specify a value for the IRLMRWT subsystem parameter (the RESOURCE TIMEOUT field on installation panel DSNTIPI). A small value can cause a large number of timeouts. With a larger value, suspended processes more often resume normally, but they remain inactive for longer periods. The default value is 30 seconds.

- If you can allow a suspended process to remain inactive for 30 seconds, use the defaults for both RESOURCE TIMEOUT and DEADLOCK TIME.
- If you specify a different a different inactive period, consider howDB2 calculates the wait time for timeouts.

Related reference:

RESOURCE TIMEOUT field (IRLMRWT subsystem parameter) (DB2 Installation and Migration)

DEADLOCK TIME field (DB2 Installation and Migration)

How DB2 calculates the wait time for timeouts

When a process requests a transaction lock that is unavailable, it waits for some period of time. DB2 determines the appropriate wait time by multiplying a timeout period by a multiplier based on the type of process.

The timeout period

PSPI

DB2 calculates a *timeout period* from the values of the RESOURCE TIMEOUT and DEADLOCK TIME options.

For example, assume that the value of the DEADLOCK TIME option is 5 and the value of the RESOURCE TIMEOUT option is 18. You can use the following calculations to see how DB2 calculates a *timeout period*.

1. Divide RESOURCE TIMEOUT by DEADLOCK TIME (18/5 = 3.6). IRLM limits the result of this division to 255.
2. Round the result to the next largest integer (Round up 3.6 to 4).
3. Multiply the DEADLOCK TIME by that integer (4 * 5 = 20).

The result, the timeout period (20 seconds), is always at least as large as the value of RESOURCE TIMEOUT (18 seconds), except when the RESOURCE TIMEOUT divided by DEADLOCK TIME exceeds 255.

The timeout multiplier

Requests from different types of processes wait for different multiples of the timeout period according to the *timeout multiplier*. In a data sharing environment, you can add another multiplier to those processes to wait for retained locks.

In some cases, you can modify the multiplier value. The following table indicates the multiplier value by type of process, and whether you can change it.

Table 12. Timeout multiplier by type

Type	Multiplier[1]	Modifiable?
IMS MPP, IMS Fast Path Message Processing, CICS, QMF, CAF, TSO batch and online, RRSAF, global transactions	1	No
IMS BMPs	4	Yes
IMS DL/I batch	6	Yes
IMS Fast Path Non-message processing	6	No
BIND subcommand processing	3	No
STOP DATABASE command processing	10	No
Utilities	6	Yes
Retained locks for all types	0	Yes

Note:

1. If the transaction occurs on a table space that is not logged, the timeout multiplier is either three or the current timeout multiplier for the thread, whichever is greater.

Changing the multiplier for IMS BMP and DL/I batch

You can modify the multipliers for IMS BMP and DL/I batch by modifying the following subsystem parameters on installation panel DSNTIPI:

IMS BMP TIMEOUT
The timeout multiplier for IMS BMP connections. A value from 1 to 254 is acceptable. The default is 4.

DL/I BATCH TIMEOUT

The timeout multiplier for IMS DL/I batch connections. A value from 1 to 254 is acceptable. The default is 6.

Additional multiplier for retained lock

For data sharing, you can specify an additional timeout multiplier to be applied to the connection's normal timeout multiplier. This multiplier is used when the connection is waiting for a retained lock, which is a lock held by a failed member of a data sharing group. A zero means don't wait for retained locks.

The scanning schedule

The following figure illustrates the following example of scanning to detect a timeout:

- DEADLOCK TIME is set to 5 seconds.
- RESOURCE TIMEOUT was chosen to be 18 seconds. Therefore, the timeout period is 20 seconds.
- A bind operation starts 4 seconds before the next scan. The operation multiplier for a bind operation is 3.

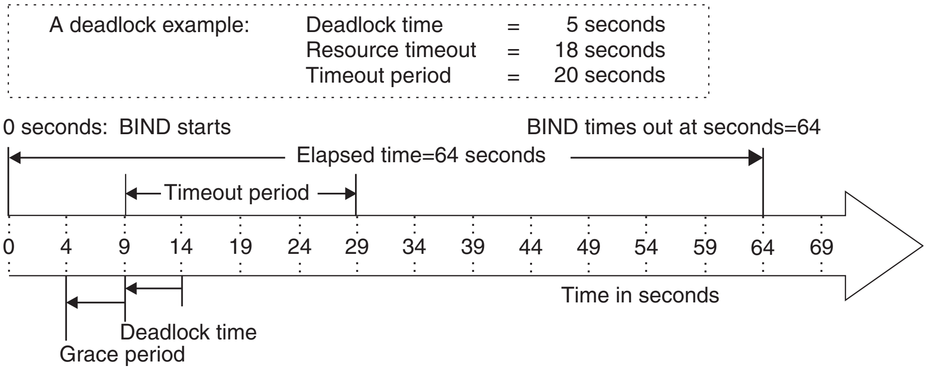


Figure 2. An example of scanning for timeout

The scans proceed through the following steps:

1. A scan starts 4 seconds after the bind operation requests a lock. As determined by the DEADLOCK TIME, scans occur every 5 seconds. The first scan in the example detects that the operation is inactive.
2. IRLM allows at least one full interval of DEADLOCK TIME as a “grace period” for an inactive process. After that, its lock request is judged to be waiting. At 9 seconds, the second scan detects that the bind operation is waiting.
3. The bind operation continues to wait for a multiple of the timeout period. In the example, the multiplier is 3 and the timeout period is 20 seconds. The bind operation continues to wait for 60 seconds longer.
4. The scan that starts 64 seconds after the bind operation detects that the process has timed out.

Consequently, an operation can remain inactive for longer than the value of RESOURCE TIMEOUT.

If you are in a data sharing environment, the deadlock and timeout detection process is longer than that for non-data-sharing systems.

You should carefully consider the length of inaction time when choosing your own values of DEADLOCK TIME and RESOURCE TIMEOUT.

PSPI

Specifying how long an idle thread can use resources

You can specify a limit for the amount of time that active distributed threads can use resources without doing any processing.

Procedure

To limit the amount of time that distributed threads can remain idle:

Specify a value other than 0 for the IDTHTOIN subsystem parameter (the IDLE THREAD TIMEOUT field on installation panel DSNTIPR) DB2 detects threads that have been idle for the specified period, and DB2 cancels the thread. Because the scan occurs only at 2-minute intervals, your idle threads generally remain idle for somewhat longer than the value you specify.
The cancellation applies only to active threads. If your installation permits distributed threads to be inactive and hold no resources, those threads are allowed to remain idle indefinitely.
The default value is 0. That value disables the scan to time out idle threads. The threads can then remain idle indefinitely.

Specifying how long utilities wait for resources

You can specify how long DB2 waits before timing out utilities that wait for locks.

Procedure

To specify the operation multiplier for utilities that wait for drain locks, transaction locks, or claims to be released:

Specify the value of the UTMOUT subsystem parameter (the UTILITY TIMEOUT field on installation panel DSNTIPI) The default value is 6. With the default value, a utility generally waits longer for a resource than does an SQL application.

Calculating wait times for drains

You can calculate how long DB2 waits for drains.

About this task

PSPI

A process that requests a drain might wait for two events:

Acquiring the drain lock.
If another user holds the needed drain lock in an incompatible lock mode, then the drainer waits.

Releasing all claims on the object.
Even after the drain lock is acquired, the drainer waits until all claims are released before beginning to process.

If the process drains more than one claim class, it must wait for those events to occur for each claim class that it drains.

Procedure

To calculate the maximum amount of wait time:

1. Add the wait time for a drain lock and the wait time for claim release. Both wait times are based on the timeout period that is calculated by DB2. For the REORG, REBUILD, REBUILD INDEX, CHECK DATA or CHECK LOB utilities, with the SHRLEVEL CHANGE options you can use utility parameters to specify the wait time for a drain lock and to indicate if additional attempts should be made to acquire the drain lock..

Drainer:	**Each wait time is:**
Utility	(timeout period) × (value of UTILITY TIMEOUT)
Other process	timeout period

2. Add the wait time for claim release.
3. Multiply the result by the number of claim classes drained.

Example

Maximum wait time: Because the maximum wait time for a drain lock is the same as the maximum wait time for releasing claims, you can calculate the total maximum wait time as follows:

For utilities

```
2
× (timeout period)
× (UTILITY TIMEOUT)
× (number of claim classes)
```

For other processes

```
2
× (timeout period)
× (operation multiplier)
× (number of claim classes)
```

For example, suppose that LOAD must drain 3 claim classes, that the timeout period is 20 seconds, and that the value of UTILITY TIMEOUT is 6. Use the following calculation to determine how long the LOAD might utility be suspended before being timed out:

```
Maximum wait time = 2
× 20
× 6
× 3 = 720 seconds
```

Wait times less than maximum: The maximum drain wait time is the longest possible time a drainer can wait for a drain, not the length of time it always waits.

For example, The following table lists the steps LOAD takes to drain the table space and the maximum amount of wait time for each step. A timeout can occur at any step. At step 1, the utility can wait 120 seconds for the repeatable read drain lock. If that lock is not available by then, the utility times out after 120 seconds. It does not wait 720 seconds.

Table 13. Maximum drain wait times: LOAD utility

Step	Maximum Wait Time (seconds)
1. Get repeatable read drain lock	120

Table 13. Maximum drain wait times: LOAD utility (continued)

Step	Maximum Wait Time (seconds)
2. Wait for all RR claims to be released	120
3. Get cursor stability read drain lock	120
4. Wait for all CS claims to be released	120
5. Get write drain lock	120
6. Wait for all write claims to be released	120
Total	**720**

PSPI

Related concepts:

How to improve performance when rebuilding index partitions (DB2 Utilities)

Related tasks:

Improving performance with REORG INDEX (DB2 Utilities)

Improving REORG TABLESPACE performance (DB2 Utilities)

Related reference:

Concurrency and compatibility for CHECK DATA (DB2 Utilities)

Concurrency and compatibility for CHECK LOB (DB2 Utilities)

Concurrency and compatibility for LOAD (DB2 Utilities)

Concurrency and compatibility for REORG INDEX (DB2 Utilities)

Concurrency and compatibility for REORG TABLESPACE (DB2 Utilities)

Chapter 7. Optimizing subsystem parameters

You can set certain subsystem parameter values to help optimize how DB2 processes SQL queries.

Procedure

PSPI

To change these values subsequently, use one of the following approaches:

- Use the SET SYSPARM command to change the parameter for the entire subsystem.
- Create statement-level optimization parameters that apply to matching SQL statements in a specified context. This method is preferred over the use of profile tables, and provides the ability to specify additional options.
- Use profile tables to specify subsystem parameter values to be used in the context of particular queries. Only certain subsystem parameters can be modified by profiles.

Results

DB2 uses the following rules to determine the precedence and scope for setting subsystem parameter values:

1. Values that are specified by statement-level or package optimization parameters. However, the existence of a valid PLAN_TABLE hint that applies to the statement prevents these values from being applied.
2. Values specified by profile tables. These parameters apply to all statements that are meet the criteria that are specified profile, and the parameters are in effect only for the time in which that profile is enabled.
3. System-level parameter values that are set by panels or the SET SYSPARM command. These parameters apply to the entire subsystem until the values are changed.

PSPI

Related tasks:

Specifying optimization parameters at the statement level

Updating subsystem parameter and application default values (DB2 Installation and Migration)

Using profiles to monitor and optimize DB2 for z/OS subsystems

Related reference:

-SET SYSPARM (DB2) (DB2 Commands)

MAX DEGREE field (PARAMDEG subsystem parameter) (DB2 Installation and Migration)

CURRENT DEGREE field (CDSSRDEF subsystem parameter) (DB2 Installation and Migration)

STAR JOIN QUERIES field (STARJOIN subsystem parameter) (DB2 Installation and Migration)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Optimizing subsystem parameters for SQL statements by using profiles

You can create profiles to specify that DB2 uses particular subsystem parameters when executing SQL statements that meet the criteria defined in the profile.

Before you begin

PSPI

Before you can use profiles to modify subsystem parameters, you must create a set of profile tables on the DB2 subsystem.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

About this task

You can use profile tables to modify the following subsystem parameters:

- NPGTHRSH
- OPTIOWGT
- STARJOIN
- SJTABLES

Procedure

To use profiles to modify the subsystem parameters that DB2 uses to execute specific SQL statements:

1. Create a profile by inserting rows in DSN_PROFILE_TABLE. The row defines the scope of the profile. The following table describes the valid combinations of scoping columns for this type of profile:

Table 14. Categories and columns used to specify valid profiles that specify subsystem parameters

Filtering category	Columns to specify
Collection identifier and package name	Specify all of the following columns: • PLANNAME (specify only '*') • COLLID • PKGNAME

2. Specify the subsystem parameter that you want to modify by inserting a row into SYSIBM.DSN_PROFILE_ATTRIBUTES with the following column values:

   PROFILE ID column
   Specify the profile that defines the scope of statements to which you want the subsystem parameter to apply. Use a value from the PROFILEID column in SYSIBM.DSN_PROFILE_TABLE.

   KEYWORDS and ATTRIBUTE*n* columns.
   Specify one of the following sets of values:

Table 15. Values for the KEYWORD and ATTRIBUTEn columns of SYSIBM.DSN_PROFILE_ATTRIBUTES

KEYWORDS column	ATTRIBUTE*n* columns
NPAGES THRESHOLD	Set ATTRIBUTE2 to an integer, 0 or greater, to specify the pages threshold for index access. (NPGTHRSH subsystem parameter.)
IO WEIGHTING	Set ATTRIBUTE1 to DISABLE or ENABLE to specify how DB2 weights I/O and CPU cost during access path selection. ENABLE is the default value. (OPTIOWGT subsystem parameter. The OPTIOWGT subsystem parameter is deprecated.)
STAR JOIN	Set ATTRIBUTE1 to DISABLE or ENABLE to specify whether DB2 uses star join processing. (STARJOIN subsystem parameter)
MIN STAR JOIN TABLES	Set ATTRIBUTE2 to an integer between 3 and 225 to specify the minimum number of tables for star join processing. (SJTABLES subsystem parameter.)

   Example: Suppose that you insert the row in the following table into SYSIBM.DSN_PROFILE_ATTRIBUTES:

Table 16. Sample data in SYSIBM.DSN_PROFILE_ATTRIBUTES.

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
17	STAR JOIN	DISABLE			2005-06-23...	

   This row specifies that DB2 is to disable star join processing for all statements that are included in profile 17.

3. Load or reload the profile tables into memory by issuing the following command:

   ```
   START PROFILE
   ```

   Any rows with a Y in the PROFILE_ENABLED column in SYSIBM.DSN_PROFILE_TABLE and SYSIBM.DSN_PROFILE_ATTRIBUTES are now in effect. DB2 applies the values for the specified parameters to the statements in the specified profile.

4. When you no longer want DB2 to use the parameter values that you specified in the profile tables, disable the values by performing one of the following actions:

Option	Description
To disable the parameter values for a specific profile	Delete that row from DSN_PROFILE_TABLE, or change the PROFILE_ENABLED column value to N. Then, reload the profile table by issuing the following command: `START PROFILE`
To disable all actions that are specified in the profile tables	Issue the following command: `STOP PROFILE`

PSPI

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

Using profiles to monitor and optimize DB2 for z/OS subsystems

Related reference:

NPGTHRSH in macro DSN6SPRM (DB2 Installation and Migration)

OPTIOWGT in macro DSN6SPRM (DB2 Installation and Migration)

STAR JOIN QUERIES field (STARJOIN subsystem parameter) (DB2 Installation and Migration)

SJTABLES in macro DSN6SPRM (DB2 Installation and Migration)

Profile tables

Chapter 8. Improving DB2 log performance

By understanding the day-to-day activity on the log, you can more effectively pinpoint when problems occur and better understand how to tune for best performance.

About this task

DB2 logs changes made to data, and other significant events, as they occur. The characteristics of your workload have a direct effect on log write performance. Long-running tasks that commit infrequently incur a lot more data to write at commit than a typical transaction. These tasks can cause subsystem impact because of the excess storage consumption, locking contention, and resources that are consumed for a rollback.

Do not forget to consider the cost of reading the log as well. The cost of reading the log directly affects how long a restart or a recovery takes because DB2 must read the log data before applying the log records back to the table space.

Related tasks:

➪ Managing the log and the bootstrap data set (DB2 Administration Guide)

Improving log write performance

By following certain recommendations, you can reduce the performance impact of writing data to the log data sets.

Procedure

To improve log write performance, use any of the following approaches:

- If you replicate your logs to remote sites, choose the storage system that provide the best possible performance for remote replication.
- Choose the largest size that your system can tolerate for the log output buffer. Because the pages for the log output buffer are permanently fixed in real storage, choose the largest size that you can dedicate in real storage. A larger size for the log output buffer might decrease the number of forced I/O operations that occur because additional buffers are unavailable, and can also reduce the number of wait conditions. You can use the OUTBUFF subsystem parameter to specify the size of the output buffer used for writing active log data sets. The maximum size of the log output buffer is 400,000 KB.

  To validate the OUTBUFF setting, you can collect IFCID 0001 (system services statistics) trace records. The QJSTWTB field indicates the number of times the buffer was full and caused a log record to wait for I/O to complete. A non-zero count for QJSTWTB might indicate that the log output buffer is too small.
- Choose fast devices for log data sets. The devices that are assigned to the active log data sets must be fast. In environments with high levels of write activity, high-capacity storage systems, such as the IBM TotalStorage DS8000 series, are recommended to avoid logging bottlenecks.
- Avoid device contention. Place the copy of the bootstrap data set and, if using dual active logging, the copy of the active log data sets, on volumes that are accessible on a path different than that of their primary counterparts.

- Preformat new active log data sets. Whenever you allocate new active log data sets, preformat them using the DSNJLOGF utility. This action avoids the overhead of preformatting the log, which normally occurs at unpredictable times.
- In most cases, do not stripe active log data sets. You can use DFSMS to the stripe the logs, but striping is generally unnecessary with the latest devices. Striping increases the number of I/Os, which can increase CPU time and lead to potentially greater DB2 commit times. Striping might improve the performance of batch insert jobs, but it might also harm the performance of online transaction processing. Striping is especially risky for performance if you replicate the logs over long distances.
- Consider striping and compressing archive log data sets by using DFSMS. Doing so might speed up the time to offload the logs and the time to recover by using archive logs. However, the performance of DFSMS striping and compression depends on the z/OS release and the types of hardware that you use.

Related concepts:

Tips for archiving with DFSMS (DB2 Administration Guide)

Related reference:

OUTPUT BUFFER field (OUTBUFF subsystem parameter) (DB2 Installation and Migration)

DSNJLOGF (preformat active log) (DB2 Utilities)

Types of log writes

Log writes are divided into two categories: asynchronous and synchronous.

Asynchronous writes

Asynchronous writes are the most common. These asynchronous writes occur when data is updated. Before, and after, image records are usually moved to the log output buffer, and control is returned to the application. However, if no log buffer is available, the application must wait for one to become available.

Synchronous writes

Synchronous writes usually occur at commit time when an application has updated data. This write is called 'forcing' the log because the application must wait for DB2 to force the log buffers to disk before control is returned to the application. If the log data set is not busy, all log buffers are written to disk. If the log data set is busy, the requests are queued until it is freed.

Writing to two logs

Dual logging is shown in the figure below.

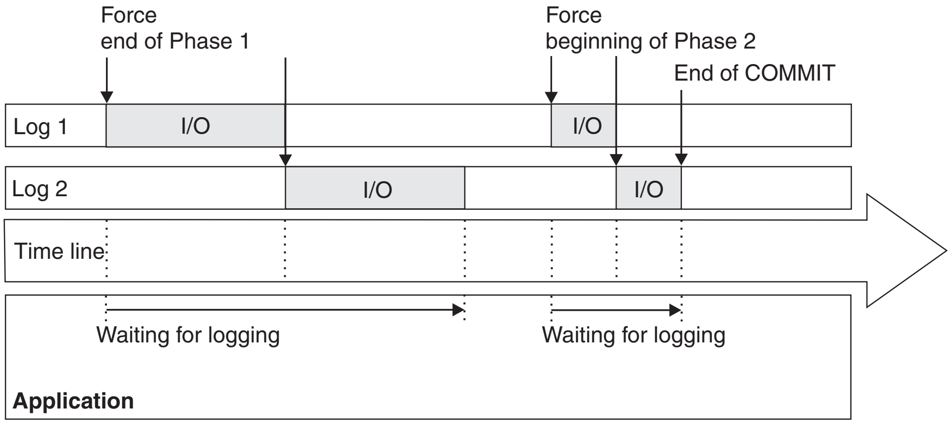


Figure 3. Dual logging during two-phase commit

If you use dual logging (recommended for availability), the write to the first log sometimes must complete before the write to the second log begins. The first time a log control interval is written to disk, the write I/Os to the log data sets are performed in parallel. However, if the same 4-KB log control interval is again written to disk, the write I/Os to the log data sets must be done serially to prevent any possibility of losing log data in case of I/O errors on both copies simultaneously.

Two-phase commit log writes

Because they use two-phase commit, applications that use the CICS, IMS, and RRS attachment facilities force writes to the log twice. The first write forces all the log records of changes to be written (if they have not been written previously because of the write threshold being reached). The second write writes a log record that takes the unit of recovery into an in-commit state.

Related reference:

DSNJLOGF (preformat active log) (DB2 Utilities)

Improving log read performance

The performance impact of log reads is evident during a rollback, restart, and database recovery.

About this task

DB2 must read from the log and apply changes to the data on disk. Every process that requests a log read has an input buffer dedicated to that process. DB2 searches for log records in the following order:

1. Output buffer
2. Active log data set
3. Archive log data set

If the log records are in the output buffer, DB2 reads the records directly from that buffer. If the log records are in the active or archive log, DB2 moves those log records into the input buffer used by the reading process (such as a recovery job or a rollback).

DB2 reads the log records faster from the active log than from the archive log. Access to archived information can be delayed for a considerable length of time if a unit is unavailable or if a volume mount is required (for example, a tape mount).

Procedure

To improve log read performance:

- Archive to disk. If the archive log data set resides on disk, it can be shared by many log readers. In contrast, an archive on tape cannot be shared among log readers. Although it is always best to avoid reading archives altogether, if a process must read the archive, that process is serialized with anyone else who must read the archive tape volume. For example, every rollback that accesses the archive log must wait for any previous rollback work that accesses the same archive tape volume to complete. If you do not have enough space to maintain the archive data sets on disk, consider using DFHSM to write the archive data sets to tape. This method has a disadvantage in that HSM must read the archive data set from disk in order to write it to tape, but the recall process is improved for a number of reasons. You can pre-stage the recalls of archive data sets in parallel (to striped data sets), and when the data sets are recalled, parallel readers can proceed.
- Avoid device contention on the log data sets by placing your active log data sets on different volumes and I/O paths to avoid I/O contention in periods of high concurrent log read activity. When multiple concurrent readers access the active log, DB2 can ease contention by assigning some readers to a second copy of the log. Therefore, for performance and error recovery, use dual logging and place the active log data sets on a number of different volumes and I/O paths. Whenever possible, put data sets within a copy or within different copies on different volumes and I/O paths. Ensure that no data sets for the first copy of the log are on the same volume as data sets for the second copy of the log.
- In most cases, do not stripe active log data sets. You can use DFSMS to the stripe the logs, but striping is generally unnecessary with the latest devices. Striping increases the number of I/Os, which can increase CPU time and lead to potentially greater DB2 commit times. Striping might improve the performance of batch insert jobs, but it might also harm the performance of online transaction processing. Striping is especially risky for performance if you replicate the logs over long distances.

Calculating average log record size

One way to determine how much log volume you need is to consider the average size of a log record.

About this task

As a general estimate, you can start with 200 bytes as the average size. To increase the accuracy of this estimate, get the real size of the log records that are written.

Procedure

To calculate the average size of log records that are written:

1. Collect the following values from the statistics report:

   LOG RECORDS CREATED

   The number of log records created (C)

   LOG CI CREATED

   The number of control intervals created in the active log counter (D)

2. Apply the following formula to the log statistics values:

```
D
× 4 KB / C  = avg size of log record
```

Example

For example, you can gather statistics for logging activities from the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report. A non-zero value for A in the following example indicates that the output buffer is too small. Ensure that the size you choose is backed up by real storage. A non-zero value for B is an indicator that the output buffer is too large for the amount of available real storage.

```
LOG ACTIVITY                 QUANTITY  /SECOND  /THREAD  /COMMIT
---------------------------  --------  -------  -------  -------
READS SATISFIED-OUTPUT BUFF      0.00     0.00      N/C     0.00
READS SATISFIED-OUTP.BUF(%)       N/C
READS SATISFIED-ACTIVE LOG       0.00     0.00      N/C     0.00
READS SATISFIED-ACTV.LOG(%)       N/C
READS SATISFIED-ARCHIVE LOG      0.00     0.00      N/C     0.00
READS SATISFIED-ARCH.LOG(%)       N/C

TAPE VOLUME CONTENTION WAIT      0.00     0.00      N/C     0.00
READ DELAYED-UNAVAIL.RESOUR      0.00     0.00      N/C     0.00
ARCHIVE LOG READ ALLOCATION      0.00     0.00      N/C     0.00
ARCHIVE LOG WRITE ALLOCAT.       0.00     0.00      N/C     0.00
CONTR.INTERV.OFFLOADED-ARCH      0.00     0.00      N/C     0.00
LOOK-AHEAD MOUNT ATTEMPTED       0.00     0.00      N/C     0.00
LOOK-AHEAD MOUNT SUCCESSFUL      0.00     0.00      N/C     0.00

UNAVAILABLE OUTPUT LOG BUFF A    0.00     0.00      N/C     0.00
OUTPUT LOG BUFFER PAGED IN B     0.00     0.00      N/C     0.00

LOG RECORDS CREATED C          9456.6K    10.5K      N/C    16.41
LOG CI CREATED D                277.3K   308.18      N/C     0.48
LOG WRITE I/O REQ (LOG1&2)      758.3K   842.70      N/C     1.32
LOG CI WRITTEN (LOG1&2)         976.8K  1085.48      N/C     1.70
LOG RATE FOR 1 LOG (MB)            N/A     2.12      N/A      N/A
```

Improving log capacity

The capacity that you specify for the active log affects DB2 performance significantly.

About this task

If you specify a capacity that is too small, DB2 might need to access data in the archive log during rollback, restart, and recovery. Accessing an archive takes a considerable amount of time.

The following subsystem parameters control the capacity of the active log. In each case, increasing the value that you specify for the parameter increases the capacity of the active log.

NUMBER OF LOGS field on installation panel DSNTIPL

Controls the number of active log data sets that you create. Having too many or too few active log data sets can each be problematic. This information is summarized in the following table.

Table 17. The effects of installation options on log data sets

Value for ARCHIVE LOG FREQ	Value for NUMBER OF LOGS	Result
Low	High	Many small data sets. Can cause operational problems when archiving to tape. Checkpoints occur too frequently.
High	Low	Few large data sets. Can result in a shortage of active log data sets.

You can use the change log inventory utility (DSNJU003) to add more active log data sets to the BSDS.

ARCHIVE LOG FREQ field on installation panel DSNTIPL
Where you provide an estimate of how often active log data sets are copied to the archive log.

UPDATE RATE field on installation panel DSNTIPL
Where you provide an estimate of how many database changes (inserts, update, and deletes) you expect per hour.

The DB2 installation CLIST
Uses UPDATE RATE and ARCHIVE LOG FREQ to calculate the data set size of each active log data set.

Certain subsystem parameters
The following subsystem parameters control the interval between checkpoints:

- CHKTYPE
- CHKFREQ
- CHKLOGR
- CHKMINS

Related concepts:

Active log data sets storage requirements (DB2 Installation and Migration)

Related tasks:

Running the installation CLIST (DB2 Installation and Migration)

Related reference:

DSNJCNVB (DB2 Utilities)

DSNJU003 (change log inventory) (DB2 Utilities)

DSNTIPB: Update selection menu panel (DB2 Installation and Migration)

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

DSNTIPL1: Checkpoint parameters (DB2 Installation and Migration)

CHECKPOINT TYPE field (CHKTYPE subsystem parameter) (DB2 Installation and Migration)

RECORDS/CHECKPOINT field (CHKFREQ and CHKLOGR subsystem parameters) (DB2 Installation and Migration)

MINUTES/CHECKPOINT field (CHKFREQ and CHKMINS subsystem parameters) (DB2 Installation and Migration)

Setting the size of active log data sets

You can modify DSNTIJIN installation job to change the size of your active log data set.

Procedure

To choose the most effective size for your active log data set:

- When you calculate the size of the active log data set, identify the longest unit of work in your application programs. For example, if a batch application program commits only once every 20 minutes, the active log data set should be twice as large as the update information that is produced during this period by all of the application programs that are running.
- Allow time for possible operator interventions, I/O errors, and tape drive shortages if offloading to tape. DB2 supports up to 20 tape volumes for a single archive log data set. If your archive log data sets are under the control of DFSMShsm, also consider the Hierarchical Storage Manager recall time, if the data set has been migrated by Hierarchical Storage Manager.
- When archiving to disk, set the primary space quantity and block size for the archive log data set so that you can offload the active log data set without forcing the use of secondary extents in the archive log data set. This action avoids space abends when writing the archive log data set.
- Make the number of records for the active log be divisible by the blocking factor of the archive log (disk or tape). DB2 always writes complete blocks when it creates the archive log copy of the active log data set. If you make the archive log blocking factor evenly divisible into the number of active log records, DB2 does not have to pad the archive log data set with nulls to fill the block. This action can prevent REPRO errors if you should ever have to REPRO the archive log back into the active log data set, such as during disaster recovery.

  To determine the blocking factor of the archive log, divide the value specified on the BLOCK SIZE field of installation panel DSNTIPA by 4096 (that is, BLOCK SIZE / 4096). Then modify the DSNTIJIN installation job so that the number of records in the DEFINE CLUSTER field for the active log data set is a multiple of the blocking factor.
- If you offload to tape, consider adjusting the size of each of your active log data sets to contain the same amount of space as can be stored on a nearly full tape volume. Doing so minimizes tape handling and volume mounts and maximizes the use of the tape resource.

  If you change the size of your active log data set to fit on one tape volume, remember that the bootstrap data set (BSDS) is copied to the tape volume along with the copy of the active log data set. Therefore, decrease the size of your active log data set to offset the space that is required on the archive tape for the BSDS.

Related concepts:

Making changes for active logs (DB2 Utilities)

Active log data sets storage requirements (DB2 Installation and Migration)

Related reference:

DSNTIPL: Active log data set parameters (DB2 Installation and Migration)

Job DSNTIJIN (DB2 Installation and Migration)

Choosing a checkpoint frequency

If log data sets are too small, checkpoints occur too frequently, and database writes are not efficient.

About this task

At least one checkpoint is taken each time that DB2 switches to a new active log data set. The recommendation is to provide enough active log space for at least 10 checkpoint intervals.

Procedure

To improve the log data set size:

Set the checkpoint frequency interval to be greater than 1 minute during peak periods In most cases, the recommendation is to specify a checkpoint interval of 2 to 5 minutes.
You can specify the interval in terms of the number of log records that are written between checkpoints or the number of minutes between checkpoints. The following subsystem parameters control the checkpoint frequency:

- CHKTYPE
- CHKFREQ
- CHKLOGR
- CHKMINS

Related concepts:

Active log data sets storage requirements (DB2 Installation and Migration)

Related tasks:

Setting the size of active log data sets

Related reference:

-SET LOG (DB2) (DB2 Commands)

-SET SYSPARM (DB2) (DB2 Commands)

DSNTIPL1: Checkpoint parameters (DB2 Installation and Migration)

CHECKPOINT TYPE field (CHKTYPE subsystem parameter) (DB2 Installation and Migration)

RECORDS/CHECKPOINT field (CHKFREQ and CHKLOGR subsystem parameters) (DB2 Installation and Migration)

MINUTES/CHECKPOINT field (CHKFREQ and CHKMINS subsystem parameters) (DB2 Installation and Migration)

Controlling the amount of log data

Certain processes such as the LOAD and REORG utility and certain SQL statements can cause a large amount of information to be logged, requiring a large amount of log space.

Controlling log size for utilities

The REORG and LOAD LOG(YES) utilities cause all reorganized or loaded data to be logged.

About this task

For example, if a table space contains 200 million rows of data, this data, along with control information, is logged when this table space is the object of a REORG utility job. If you use REORG with the DELETE option to eliminate old data in a table and run CHECK DATA to delete rows that are no longer valid in dependent tables, you can use LOG(NO) to control log volume.

Procedure

To reduce the log size:

- When populating a table with many records or reorganizing table spaces or indexes, specify LOG(NO) and take an inline copy or take a full image copy immediately after the LOAD or REORG.
- Specify LOGGED when adding less than 1% of the total table space. Doing so creates additional logging, but eliminates the need for a full image copy

Controlling log size for SQL operations

The amount of logging performed for applications depends on how much data is changed.

About this task

Certain SQL statements are quite powerful, and a single statement can sometimes modify a large amount of data. Such statements include:

INSERT with a fullselect
A large amount of data can be inserted into table or view, depending on the result of the query.

Mass deletes and mass updates (except for deleting all rows for a table in a segmented or universal table space)

For non-segmented table spaces, each of these statements results in the logging of all database data that changes. For example, if a table contains 200 million rows of data, that data and control information are logged if all of the rows of a table are deleted with the SQL DELETE statement. No intermediate commit points are taken during this operation.

For segmented and universal table spaces, a mass delete results in the logging of the data of the deleted records when any of the following conditions are true:

- The table is the parent table of a referential constraint.
- The table is defined as DATA CAPTURE(CHANGES), which causes additional information to be logged for certain SQL operations.
- A delete trigger is defined on the table.

TRUNCATE TABLE
Essentially a mass-delete that does not activate delete triggers

Data definition statements
Logging for the entire database descriptor for which the change was made. For very large DBDs, this can be a significant amount of logging.

Modification to rows that contain LOB or XML data
Data in tables that contain LOB or XML data.

Procedure

To control the use of log space by powerful SQL statements:

- For mass delete operations, consider using segmented table spaces or universal table spaces. If segmented table spaces are not an option, and no triggers exist on the table or your application can safely ignore any triggers on the table, create one table per table space, and use TRUNCATE.
- For inserting a large amount of data, instead of using an SQL INSERT statement, use the LOAD utility with LOG(NO) and take an inline copy.
- For updates, consider your workload when defining a table's columns. The amount of data that is logged for update depends on whether the row contains all fixed-length columns or not. For fixed-length non-compressed rows, changes are logged only from the beginning of the first updated column to the end of the last updated column. Consequently, you should keep frequently updated columns close to each other to reduce log quantities.

  For varying-length rows (A varying-length row contains one or more varying-length columns), data is logged from the first changed byte to the end of the last updated column if the length is not changed. However, in cases where the length changes, which are more common, the data is logged from the first changed byte to the end of the row.

  To determine whether a workload is read-intensive or update-intensive, check the log data rate. You can find the rate in the LOG RATE FOR 1 LOG (MB/SEC) field in the log statistics. Determine the average log size and divide that by 60 to get the average number of log bytes written per second.
  - If you log less than 2 MB per second, the workload is read-intensive.
  - If you log more than 20 MB per second, the workload is update-intensive.
  - From 2 - 20 MB per second, the workload is neither read- or update-intensive.
- If you have many data definition statements (CREATE, ALTER, DROP) for a single database, issue them within a single unit of work to avoid logging the changed DBD for each data definition statement. However, be aware that the DBD is locked until the COMMIT is issued.
- Use the NOT LOGGED option for any LOB or XML data that requires frequent updating and for which the trade off of non-recoverability of LOB or XML data from the log is acceptable. (You can still use the RECOVER utility on LOB or XML table spaces to recover control information that ensures physical consistency of the LOB or XML table space.) Because LOB and XML table spaces defined as NOT LOGGED are not recoverable from the DB2 log, you should make a recovery plan for that data. For example, if you run batch updates, be sure to take an image copy after the updates are complete.
- For data that is modified infrequently, except during certain periods such as year-end processing, when frequent or large changes to the data occur over a short time:
  1. Make an image copy of the data.
  2. Alter the table space to NOT LOGGED.
  3. Make the massive changes.
  4. Stop other activities that update the data.
  5. Make an image copy of the data.
  6. Alter the table space to LOGGED.
- For changes to tables, such as materialized query tables, that contain propagated data, use the NOT LOGGED option because the data exists elsewhere. If the data becomes damaged you can refresh the entire table from the original source.

Related concepts:

- The NOT LOGGED attribute (DB2 Administration Guide)

Related tasks:

- Managing the log (DB2 Administration Guide)

Related reference:

- LOAD (DB2 Utilities)
- RECOVER (DB2 Utilities)
- INSERT (DB2 SQL)
- TRUNCATE (DB2 SQL)

Chapter 9. Managing DB2 threads

You can specify maximum numbers of active allied and database access threads that can be allocated concurrently.

About this task

Threads are an important resource within a DB2 subsystem. A *thread* is a structure that describes a connection made by an application and traces its progress in the DB2 subsystem. These values are specified at installation, and controlled by subsystem parameters. Choosing appropriate values for the maximum numbers of threads is important to keep applications from queuing, and to provide adequate response times.

Related tasks:

"Monitoring threads and connections by using profiles" on page 108

Monitoring threads (DB2 Administration Guide)

Types of threads

Threads are an important resource within a DB2 subsystem. A *thread* is a structure that describes a connection made by an application and traces its progress in the DB2 subsystem.

Allied threads
Allied threads are threads that are connected to DB2 from TSO, batch, IMS, CICS, CAF, or RRSAF.

Database Access Threads
Distributed *database access threads* (sometimes called DBATs) are threads that are connected through a network to access data at a DB2 server on behalf distributed requesting systems. Database access threads are created in the following situations:

- When new connections are accepted from remote requesters
- When DB2 is configured in INACTIVE mode, and a new request is received from a remote requester and no pooled database access thread is available to service the new request

Database access threads can operate in ACTIVE or INACTIVE mode. The mode used for database access threads is controlled by the value of the CMTSTAT subsystem parameter.

INACTIVE mode
When the value of the CMTSTAT subsystem parameter is INACTIVE, a database access thread can be active or pooled. When a database access thread is *active*, it is processing requests from client connections within units of work. When a database access thread is *pooled*, it is waiting for the next request from a client to start a new unit of work.

INACTIVE mode database access threads are terminated under any of the following conditions:

- After processing 200 units of work.

- After being idle in the pool for the amount of time specified by the value of the POOLINAC subsystem parameter.

However, the termination of an INACTIVE mode thread does not prevent another database access thread from being created to meet processing demand, as long as the value of the MAXDBAT subsystem parameter has not been reached.

ACTIVE mode
When the value CMTSTAT subsystem parameter is ACTIVE, a database access thread is always active from initial creation to termination.

Related reference:

DDF THREADS field (CMTSTAT subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

IDLE THREAD TIMEOUT field (IDTHTOIN subsystem parameter) (DB2 Installation and Migration)

How DB2 allocates allied threads

This information describes at a high level the steps that DB2 uses for the allocation of an allied thread. It also describes some of the factors related to the performance of those steps.

This information does not apply to the allocation of database access threads (DBATs).

1. Thread creation

The following list shows the main steps in thread creation.

1. DB2 checks whether the maximum number of active threads has been exceeded. The thresholds are specified in the CTHREAD subsystem parameter or MAX REMOTE MAXDBAT subsystem parameter. If it has been exceeded, the request waits. The wait for threads is not traced, but the number of requests queued is provided in the performance trace record with IFCID 0073.
2. DB2 checks the authorization ID for an application plan in the SYSIBM.SYSPLANAUTH catalog table (IFCID 0015). If this check fails, the table SYSUSERAUTH is checked for the SYSADM special privilege.
3. For application plans, DB2 loads the control structures associated with the plan. The control block for an application plan is divided into sections. The header and directory contain control information; SQL sections contain SQL statements from the application. A copy of the control structure for the plan is made for each thread executing the plan. Only the header and directory are loaded when the thread is created.
4. DB2 loads the descriptors necessary to process the plan. Some of the control structures describe the DB2 table spaces, tables, and indexes used by the application.

2. Resource allocation

Some of the structures necessary to process the statement are stored in 4 KB pages. If they are not already present, those structures are read into database buffer pool BP0 and copied from there into the EDM pool.

1. If it is not already in the EDM pool, DB2 loads the section of the control structure that corresponds to the SQL statement.
2. DB2 loads structures that are referred to by the SQL statement that are not already in the EDM pool.
3. Allocate and open data sets. When the control structure is loaded, DB2 locks the resources used.

3. SQL statement execution

If the statement resides in a package, the directory and header of the control structure for the package are loaded at the time of the first execution of a statement in the package.

The control structure for the package is allocated at statement execution time. The header of the control structure for the plan is allocated at thread creation time.

When the package is allocated, DB2 uses the package authorization cache or the SYSPACKAUTH catalog table checks authorization . DB2 checks to see that the plan owner has execute authority on the package. On the first execution, the information is not in the cache; therefore, the catalog is used. After the first execution, the cache is used.

For dynamic bind, authorization checking also occurs at statement execution time.

A summary record, produced at the end of the statement (IFCID 0058), contains information about each scan that is performed.

From a system performance perspective, the most important factor in the performance of SQL statement execution is the size of the database buffer pool. If the buffer pool is large enough, some index and data pages can remain there and can be accessed again without an additional I/O operation.

4. Commit and thread termination

Commit processing can occur many times while a thread is active. For example, an application program running under the control structure of the thread could issue an explicit COMMIT or SYNCPOINT several times during its execution. When the application program or the thread terminates, an implicit COMMIT or SYNCPOINT is issued.

When a COMMIT or SYNCPOINT is issued from an IMS application running with DB2, the two-phase commit process begins if DB2 resources have been changed since the last commit point. In a CICS or RRSAF application, the two-phase commit process begins only if DB2 resources have changed and a resource outside of DB2 has changed within the same commit scope.

The significant events that show up in a performance trace of a commit and thread termination operation occur in the following sequence:

1. In commit phase 1 (IFCID 0084), DB2 writes an end of phase 1 record to the log (IFCIDs 0032 and 0033). The trace shows two I/O operations, one to each active log data set (IFCIDs 0038 and 0039).
2. In commit phase 2 (IFCID 0070), DB2 writes a beginning of phase 2 record to the log. Again, the trace shows two I/O operations. Page and row locks , held to a commit point, are released. An unlock (IFCID 0021) with a requested token of zeros frees any lock for the specified duration. A summary lock record (IFCID 0020) is produced, which gives the maximum number of page locks held and the number of lock escalations. DB2 writes an end of phase 2 record to the log.

   If RELEASE(COMMIT) is used, the following events also occur:

   - Table space locks are released.
   - All the storage used by the thread is freed, including storage for control blocks, CTs and PTs, and working areas.
   - The use counts of the DBDs are decreased by one. If space is needed in the EDM DBD cache, a DBD can be freed when its use count reaches zero.
   - Those table spaces and index spaces with no claimers are made candidates for deferred close.
3. The thread is terminated, and the accounting record is written. The accounting record does not report transaction activity that takes place before the thread is created.

   If RELEASE(DEALLOCATE) is used to release table space locks, the DBD use count is decreased, and the thread storage is released.

Related tasks:

Choosing a RELEASE option

Related reference:

DSNTIPE: Thread management panel (DB2 Installation and Migration)

MAX USERS field (CTHREAD subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

Trace field descriptions

Reusing threads

The cost of creating a thread can be significant and reusing threads is a way to avoid that cost.

About this task

In general, you want transactions to reuse threads when transaction volume is high and the cost of creating threads is significant, but thread reuse is also useful for a lower volume of priority transactions. For a transaction of five to ten SQL statements (10 I/O operations), the cost of thread creation can be 10% of the processor cost. But the steps needed to reuse threads can incur costs of their own.

Procedure

To reduce the costs of creating many threads, use the following approaches:

- For allied threads, bind plans with the RELEASE(DEALLOCATE) option.

RELEASE(DEALLOCATE) does not free cursor tables (SKCTs) at a commit point. Therefore, the cursor table could grow as large as the plan. If you are using created temporary tables, the logical work file space is not released until the thread is deallocated. Thus, many uses of the same created temporary table do not cause reallocation of the logical work files, but be careful about holding onto this resource for long periods of time if you do not plan to use it.

- For database access threads, enabled threads to be pooled at the DB2 server As a server, DB2 can assign that connection to a pooled database access thread. Those threads can be shared and reused by thousands of client connections, which lets DB2 support very large client networks at minimal cost. (Inactive threads are only eligible to be reused by the same connection.)

  If your server is not DB2 for z/OS, or some other server that can reuse threads, then reusing threads for your requesting CICS, IMS, or RRS applications is not a benefit for distributed access. Thread reuse occurs when sign-on occurs with a new authorization ID. If that request is bound for a server that does not support thread reuse, that change in the sign-on ID causes the connection between the requester and server to be released so that it can be rebuilt again for the new ID.

Related tasks:

Enabling distributed database access threads to be pooled

Choosing a RELEASE option

Related reference:

RELEASE bind option (DB2 Commands)

Analyzing the reuse of threads

The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report can help you identify, by plan, when threads were reused.

About this task

The following sections of the report contain information about the thread reuse:

- NORMAL TERM.
- ABNORMAL TERM.
- IN DOUBT

Example

The following example reports shows the location of the following values:

- NEW USER (A) tells how many threads were not terminated at the end of the previous transaction or query, and hence reused.
- DEALLOCATION (B) tells how many threads were terminated at the end of the query or transaction.
- APPL. PROGR. END (C) groups all the other reasons for accounting. Since the agent did not abend, these are considered normal terminations.

```
NORMAL TERM.      TOTAL  ABNORMAL TERM.      TOTAL  IN DOUBT            TOTAL
----------------  -----  ------------------  -----  -----------------   ------

NEW USER  A           0  APPL.PROGR. ABEND       0  APPL.PGM. ABEND          0
DEALLOCATION  B       0  END OF MEMORY           0  END OF MEMORY            0
APPL.PROGR. END  C    0  RESOL.IN DOUBT          0  END OF TASK              0
RESIGNON  D           0  CANCEL FORCE            0  CANCEL FORCE             0
DBAT INACTIVE           0
TYPE2 INACTIVE        193
```

```
RRS COMMIT           0
END U. THRESH        0
BLOCK STORAGE        0
STALENESS            0
```

This technique is accurate in IMS but not in CICS, where threads are reused frequently by the same user. For CICS, also consider looking at the number of commits and aborts per thread. For CICS:

- NEW USER (A) is thread reuse with a different authorization ID or transaction code.
- RESIGN-ON (D) is thread reuse with the same authorization ID.

Enabling distributed database access threads to be pooled

You can enable distributed database access threads to be pooled to achieve certain performance advantages.

About this task

Enabling threads to be pooled provides the following advantages:

- You can leave an application that is running on a workstation connected to DB2 from the time the application is first activated until the workstation is shut down and thus avoid the delay of repeated connections.
- DB2 can support a larger number of distributed connections limited by the value of CONDBAT subsystem parameter, because the number of connections is not limited by the maximum number of threads.
- Less storage is used for each DDF connection because a connection uses significantly less storage than a database access thread.
- You get an accounting trace record each time that a thread is pooled rather than once for the entire time that you are connected. When a pooled thread becomes active, the accounting fields for that thread are re-initialized. As a result, the accounting record contains information about active threads only, which makes it easier to study how distributed applications are performing. If a pooled mode thread remains active because of sections that are specified with the KEEPDYNAMIC(YES) bind option, an accounting record is still written.

  Exception: If you employ account record accumulation, an accounting trace is not produced each time that a thread becomes pooled. Accounting records can be rolled up by concatenated combinations of the following values:
  - User ID
  - End transaction name
  - User workstation name
- Each time that a thread is pooled, Workload Manager resets the information that it maintains on that thread. The next time that thread is activated, workload manager begins managing to the goals that you have set for the transactions that run in that service class. If you use multiple performance periods, it is possible to favor short-running units of work that use fewer resources while giving fewer resources over time to long running units of work.
- You can use response time goals, which is not recommended when ACTIVE mode threads are used.
- INACTIVE mode threads can better take advantage of the ability to time out idle active threads.
- Thread reuse enables DDF to use a small pool of database access threads to support a large group of network clients.

- The response times reported by RMF include periods between requests and within the same unit of work. These times are shown as idle delays.

Procedure

To enable distributed database access threads to be pooled:

Set the value of the CMTSTAT subsystem parameter to INACTIVE. The recommendation is to use INACTIVE mode for database access threads instead of ACTIVE mode, whenever possible.

Results

DB2 always tries to pool database access threads, but in some cases such threads cannot be pooled. The conditions listed in the following table determine whether a thread can be pooled. When the conditions are true, the thread can be pooled when a COMMIT is issued.

Table 18. Requirements for pooled threads

If there is...	Thread can be pooled?
A DRDA hop to another location with a pending request[1]	No
A package that is bound with RELEASE(COMMIT)	Yes
A package that is bound with RELEASE(DEALLOCATE)	Yes
A held cursor, a held LOB locator, or a package bound with KEEPDYNAMIC(YES)	No
A declared temporary table that is active (the table was not explicitly dropped through the DROP TABLE statement or the ON COMMIT DROP TABLE clause on the DECLARE GLOBAL TEMPORARY TABLE statement)	No

[1] A pending request occurs when a hop connection is created but no SQL statements are run at that location.

After a ROLLBACK, a thread can be pooled even if it had open cursors defined WITH HOLD or a held LOB locator because ROLLBACK closes all cursors and LOB locators. ROLLBACK is also the only way to use the KEEPDYNAMIC(YES) bind option to clear information about a dynamic SQL statement.

Related tasks:

Reusing threads

Establishing performance periods for DDF threads

Timing out idle active threads

Related reference:

DDF THREADS field (CMTSTAT subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE CONNECTED field (CONDBAT subsystem parameter) (DB2 Installation and Migration)

COMMIT (DB2 SQL)

ROLLBACK (DB2 SQL)

RELEASE bind option (DB2 Commands)

Setting thread limits

You can limit the maximum number of DB2 threads that can be allocated concurrently.

About this task

Set these values to provide good response time without wasting resources, such as virtual and real storage. The value that you specify depends on your machine size, your workload, and other factors. When specifying values for these subsystem parameters, consider the following factors:

- Fewer threads than needed under-use the processor and cause the queuing of threads.
- More threads than needed do not improve the response time. They require more real storage for the additional threads and might cause more paging, and therefore, performance degradation.

Procedure

To limit the number of allied and database access threads that can be allocated concurrently:

- Specify a value for the CTHREAD subsystem parameter to control the maximum number of concurrent allied threads that are started at the local DB2 subsystem.
- Specify a value for the MAXDBAT subsystem parameter to control the maximum number of concurrently active database access threads. The sum of the values of the CTHREAD and MAXDBAT subsystem parameters is limited to a maximum value of 20000.
- For the TSO and call attachment facilities, you can limit the number of threads indirectly by specifying the values of the IDFORE and IDBACK subsystem parameters. These values limit the number of connections to DB2. The number of threads and connections that are allowed affects the amount of work that DB2 can process.

Related reference:

MAX USERS field (CTHREAD subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

MAX TSO CONNECT field (IDFORE subsystem parameter) (DB2 Installation and Migration)

MAX BATCH CONNECT field (IDBACK subsystem parameter) (DB2 Installation and Migration)

DSNTIPE: Thread management panel (DB2 Installation and Migration)

Setting thread limits for database access threads

You can specify the maximum number of concurrently active allocated threads within the DB2 subsystem.

Procedure

To control the maximum number of concurrently allocated active threads, use the following approaches:

- Specify the value of the CTHREAD subsystem parameter to control the maximum number of allied threads that can be concurrently allocated. The sum of the values of the CTHREAD and MAXDBAT subsystem parameters must not exceed 20000.
- Specify the value of the CONDBAT subsystem parameter to control the concurrent inbound DDF connections. The maximum acceptable value for CONDBAT subsystem parameter is 150000. However, this upper limit is obtained only if you specify INACTIVE for value of the CMTSTAT subsystem parameter.

Related tasks:

"Monitoring threads and connections by using profiles" on page 108

Setting thread limits

"Setting limits for the queuing of client connections waiting for database access threads" on page 106

Related reference:

DDF THREADS field (CMTSTAT subsystem parameter) (DB2 Installation and Migration)

MAX USERS field (CTHREAD subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE CONNECTED field (CONDBAT subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

Timing out idle active threads

You can specify a time limit for active threads to remain idle.

About this task

When a thread remains idle than the specified limit, DB2 might cancel the thread. However, pooled and in-doubt threads are not canceled.

Procedure

To specify limits for active idle threads:

- Set the value of the IDTHTOIN subsystem parameter. The timeout period is an approximation. If a server thread has been waiting for a request from the requesting site for this period of time, it is canceled unless the thread is currently pooled or in doubt thread. A value of 0, the default, means that the server threads cannot be canceled because of an idle thread timeout. You can specify a value from 0 to 9999 seconds.
- Set the value of the CMTSTAT subsystem parameter to ACTIVE. When you specify ACTIVE, as application must start its next unit of work within the specified timeout period, otherwise its thread is canceled.

- Set a value for the TCPKPALVsubsystem parameter. A TCP/IP keep alive interval of 5 minutes or less, in conjunction with an IDTHTOIN value, can ensure that resources are not locked for a long time when a network outage occurs.

Related tasks:

"Monitoring idle threads by using profiles" on page 115

Related reference:

DDF THREADS field (CMTSTAT subsystem parameter) (DB2 Installation and Migration)

IDLE THREAD TIMEOUT field (IDTHTOIN subsystem parameter) (DB2 Installation and Migration)

TCP/IP KEEPALIVE field (TCPKPALV subsystem parameter) (DB2 Installation and Migration)

Setting limits for the queuing of client connections waiting for database access threads

You can establish limits for the depth of the client connection queue and the time that client connections wait for database access threads (DBATs).

Before you begin

This function is supported only when the DB2 subsystem is a member of a data sharing group.

About this task

You can specify thresholds to control the queuing of connections waiting for DBATs. When the thresholds are reached, some connections are closed. The closing of connections provides opportunities for remote client systems to redirect work to other members of the data sharing group that have more resources.

Procedure

To control queuing for client connections, use the following approaches:

- Specify the value of the MAXCONQN subsystem parameter to control the maximum number of connections in the queue waiting for a DBAT. You can apply either of the following choices:
  - Set the value to ON to limit the number of queued connections to the value of the MAXDBAT subsystem parameter.
  - Specify a numeric value to indicate the threshold for the number of queued connections. The value that you specify must be less than or equal to the maximum supported value for the MAXDBAT subsystem parameter.
- Specify the value of the MAXCONQW subsystem parameter to control the maximum time that connections wait for DBATs. You can apply either of the following choices:
  - Set the value to ON to limit the amount of time that queued connections wait for DBATs to the value of the IDTHTOIN subsystem parameter.
  - Specify a numeric value to limit the amount of time (in seconds) that queued connections wait for DBATs. You can specify a value equal to or greater than 5 seconds and less than or equal to 3600 seconds.

Related tasks:

Setting thread limits for database access threads

Related reference:

MAX REMOTE CONNECTED field (CONDBAT subsystem parameter) (DB2 Installation and Migration)

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

IDLE THREAD TIMEOUT field (IDTHTOIN subsystem parameter) (DB2 Installation and Migration)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Controlling allocation and deallocation processing for database access threads

You can use package release options to reduce the use of CPU resources for package allocation and deallocation processing and retain the ability to conduct routine and emergency maintenance tasks.

About this task

When database access threads run in INACTIVE mode according to the value of the CMTSTAT subsystem parameter, DB2 honors the RELEASE bind option of the package. When packages use the RELEASE(DEALLOCATE) option, the amount of allocation and deallocation processing is reduced, because the copy of the package remains allocated until the database access thread is terminated.

However, database access threads that run according the rules of the RELEASE(DEALLOCATE) bind option can hold package locks and table space intent locks that can effectively block your ability to modify objects, invoke utilities, or bind or rebind packages. The MODIFY DDF command enables you to change the processing rules that DB2 uses for database access threads.

Procedure

To minimize the use of CPU resources for package allocation and deallocation processing, use the following practices:

- During normal production operating hours, enable DB2 to allocate and deallocate packages according to the rules of the RELEASE(DEALLOCATE) bind option.
  - You can issue either of the following commands to specify that DB2 uses the rules of the RELEASE bind option that is specified for the package:

    Specify that terminated DBATs are deallocated
    ```
    MODIFY DDF PKGREL(BNDOPT)
    ```

    Specify that terminated DBATs are pooled
    ```
    MODIFY DDF PKGREL(BNDPOOL)
    ```
- For nightly utilities and emergency maintenance, specify that DB2 uses the rules of the RELEASE(COMMIT) bind option for all packages. You can issue the following command to specify that behavior for all packages, regardless of the RELEASE option specified for each package:
  ```
  MODIFY DDF PKGREL(COMMIT)
  ```

The effects of issuing this command are not immediate. However, any database access thread is terminated and made inactive when the next commit point is reached. Any RELEASE(DEALLOCATE) database access threads that remain active waiting for a new unit-of-work request from a client are terminated by a DDF service task, which runs every two minutes. Subsequently, when any new unit-of-work creates a database access thread, the packages are allocated according to the rules of the RELEASE(COMMIT) bind option.

Related tasks:

Choosing a RELEASE option

Related reference:

RELEASE bind option (DB2 Commands)

-MODIFY DDF (DB2) (DB2 Commands)

Monitoring threads and connections by using profiles

You can monitor threads and connections for remote TCP/IP access to DB2 servers. You can use the resulting information to analyze the use of system resources by particular clients, applications, and users. You can also create exception thresholds to prioritize resources accordingly.

Before you begin

Create a complete set of profile tables on the DB2 subsystem.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:
- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

Procedure

1. Populate the profile table rows with values that specify the monitoring function, the type of monitoring (warning or exception), and the thresholds:
   - Monitoring connections by using profiles
   - Monitoring threads by using profiles
   - Monitoring idle threads by using profiles
2. Issue the START PROFILE command to load or reload the profile tables into memory and start the specified monitoring functions.

What to do next

When you are finished monitoring threads and remote connections, stop the profile by taking one of the following actions:
- To disable monitoring for a specific profile:

1. Delete the row for the profile from the SYSIBM.DSN_PROFILE_TABLE table, or change the value of the PROFILE_ENABLED column to 'N'.
2. Reload the profile table into memory by issuing the START PROFILE command.

- To disable all monitoring, issue the STOP PROFILE command.

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

“Using profiles to monitor and optimize DB2 for z/OS subsystems” on page 657

Setting thread limits for database access threads

Related reference:

Profile tables

-START PROFILE (DB2) (DB2 Commands)

-STOP PROFILE (DB2) (DB2 Commands)

Related information:

DSNT771I (DB2 Messages)

DSNT772I (DB2 Messages)

-30041 (DB2 Codes)

Monitoring connections by using profiles

You can monitor connections for remote TCP/IP access to DB2 servers. You can use the resulting information to analyze the use of system resources by particular clients, applications, and users. You can also create exception thresholds to prioritize resources accordingly.

Before you begin

Before you can use profiles to monitor threads and connections, create a complete set of profile tables on the DB2 subsystem.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

Procedure

To monitor remote connections.

1. Insert values in the SYSIBM.DSN_PROFILE_TABLE table to create the profile and specify the filtering scope of the profile:
   a. Insert a value in the PROFILEID column to identify the profile.

b. Insert values to specify the filtering criteria: You can specify only IP address or domain name values in the LOCATION column. Other combinations of criteria are not accepted for this monitoring function.

2. Inserting rows into the SYSIBM.DSN_PROFILE_ATTRIBUTES table to specify the monitoring function of the profile, the type of monitoring and the threshold to monitor:

   a. Insert a value in the PROFILEID column to identify which profiles use this type monitoring function. Insert a value that matches value of the PROFILEID column in the corresponding SYSIBM.DSN_PROFILE_TABLE rows.

   b. Specify the monitoring function by inserting the 'MONITOR CONNECTIONS' value in the KEYWORD column.

      The profile monitors the total number of remote connections from TCP/IP requesters, including the current active connections and the inactive connections. Active connections are either currently associated with an active database access thread or have been queued and are waiting to be serviced. Inactive connections are currently not waiting and not associated with a database access thread.

      This monitoring function is subject only to filtering by the LOCATION column in the SYSIBM. DSN_PROFILE_TABLE. However, you can specify either an IP address or a domain name for the value the LOCATION column.

      The system-wide threshold that is defined by the value of the CONDBAT subsystem parameter continues to apply. Because the threshold specified by the subsystem parameter would always apply first, DB2 rejects any profile that specifies a threshold for the MONITOR CONNECTIONS keyword that is higher than the value of the CONDBAT subsystem parameter.

   c. Insert values in the ATTRIBUTE1 column of DSN_PROFILE_ATTRIBUTES table to specify how DB2 responds when a threshold is met. The possible values in the ATTRIBUTE1 column has two parts:

      - The first part is the type of monitoring:

        WARNING
        A console message is issued at most every 5 minutes depending on the specified diagnosis level. When WARNING is specified, WARNING_DIAGLEVEL1 is used.

        EXCEPTION
        DB2 issues the messages that are specified by the diagnosis level and rejects any new incoming remote connection requests.

      - The second part of the value in the ATTRIBUTE1 column controls the amount of detail in the message that DB2 issues when the threshold is met:

        ***type*_DIAGLEVEL1**
        Where *type* is WARNING or EXCEPTION, DB2 takes the appropriate specified action and issues a DSNT771I console message, which contains minimal information.

        ***type*_DIAGLEVEL2**
        Where *type* is WARNING or EXCEPTION, DB2 takes the specified action. DB2 also issues a DSNT772I console message, which contains additional information such as the profile ID and reason code in the message text.

d. Insert values in the ATTRIBUTE2 column of DSN_PROFILE_ATTRIBUTES table to specify the threshold that the monitor uses.

The following table summarizes the meanings of the different columns of the DSN_PROFILE_ATTRIBUTES table for the MONITOR CONNECTIONS .

Table 19. Summary of DSN_PROFILE_ATTRIBUTES values for monitoring threads and connections

KEYWORD	ATTRIBUTE1 column	ATTRIBUTE2 column	ATTRIBUTE3 column
MONITOR CONNECTIONS	Specify one of the following types of monitoring: • WARNING • WARNING_DIAGLEVEL1 • WARNING_DIAGLEVEL2 • EXCEPTION • EXCEPTION_DIAGLEVEL1 • EXCEPTION_DIAGLEVEL2	Insert a value to indicate the threshold for the maximum allowed number of remote connections that meet the profile criteria. The value that you specify must be less than or equal to the value of the CONDBAT subsystem parameter.	Not used.

3. Issue the START PROFILE command to start the monitoring functions that are specified in the profile tables.

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related reference:

-START PROFILE (DB2) (DB2 Commands)

SYSIBM.DSN_PROFILE_TABLE

SYSIBM.DSN_PROFILE_ATTRIBUTES

MAX REMOTE CONNECTED field (CONDBAT subsystem parameter) (DB2 Installation and Migration)

Related information:

DSNT771I (DB2 Messages)

DSNT772I (DB2 Messages)

00E30503 (DB2 Codes)

00E30504 (DB2 Codes)

Monitoring threads by using profiles

You can monitor threads for remote TCP/IP access to DB2 servers. You can use the information to analyze the use of system resources by particular clients, applications, and users, and prioritize resources accordingly.

Before you begin

Create a complete set of profile tables on the DB2 subsystem.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY

- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

Procedure

To monitor threads:

1. Insert values in the SYSIBM.DSN_PROFILE_TABLE table to create and specify the scope of the profile:
   a. Insert a value in the PROFILEID column to identify the profile.
   b. Insert values to specify the filtering criteria: You can specify filtering criteria from any one of the following combinations:
      1) IP address or domain name, in the LOCATION column.
      2) Product identifier, in the PRDID column.
      3) Role and authorization identifier, in both ROLE and AUTHID columns.
      4) Role, in the ROLE column only.
      5) Authorization identifier, in the AUTHID column only.
      6) Server location name, location alias, or database name, in the LOCATION column.
      7) The location name of a requester, for monitored threads from a DB2 for z/OS requester.
      8) Collection identifier and package name, in both COLLID and PKGNAME columns.
      9) Collection identifier, in the COLLID column only.
      10) Package name, in the PKGNAME column only.
      11) Client application name, in the CLIENT_APPLNAME column.
      12) Client user identifier, in the CLIENT_USERID column.
      13) Client workstation name, in the CLIENT_WRKSTNNAME column.

      Other combinations of criteria are not accepted. You can specify several profiles that have overlapping criteria. DB2 might apply such overlapping profiles in combination, or it might apply one profile and exclude others, depending on the overlapping criteria.
2. Specify the monitoring function of the profile by inserting rows into the SYSIBM.DSN_PROFILE_ATTRIBUTES table.
   a. Insert a value in the PROFILEID column to identify which profiles use this type monitoring function. Insert a value that matches value of the PROFILEID column in the corresponding SYSIBM.DSN_PROFILE_TABLE rows.
   b. Insert the 'MONITOR THREADS' value in the KEYWORD column of DSN_PROFILE_ATTRIBUTES table to specify the monitoring function.

      The profile monitors the total number of concurrent active remote threads that use TCP on the DB2 subsystem.

      When the profile criteria are based on only the collection or package, DB2 checks only the first collection or package that is associated the first SQL statement that is executed under the thread.

      The system-wide threshold that is defined by the value of the MAXDBAT subsystem parameter continues to apply. Therefore DB2 rejects any profile that specifies a threshold for the MONITOR THREADS keyword that is higher than the value of the MAXDBAT subsystem parameter.

c. Insert values in the ATTRIBUTE1 column of DSN_PROFILE_ATTRIBUTES table to specify how DB2 responds when a threshold is met.

d. Insert values in the ATTRIBUTE2 column of DSN_PROFILE_ATTRIBUTES table to specify the threshold that the monitor uses.

The possible values in the ATTRIBUTE1 column have two parts. The first part is the type of monitoring:

WARNING
A console message is issued at most every 5 minutes depending on the specified diagnosis level. When WARNING is specified, WARNING_DIAGLEVEL1 is used.

EXCEPTION
When EXCEPTION is specified, EXCEPTION_DIAGLEVEL1 is used. If the profile threshold is zero and is exceeded, the thread is canceled. When the profile threshold is greater than zero, the action taken depends on the filtering scope of the profile as described in the following table:

Table 20. Actions taken when thresholds for EXCEPTION profiles are exceeded

Filtering scope	Action
IP Address or domain name	The thread is queued until another server thread becomes available (that is, when another thread deallocates (in ACTIVE mode) or becomes inactive (in INACTIVE mode)) and the number of concurrent active threads falls below the profile exception threshold. The connection that is associated with the thread remains open and is not terminated.
Product identifier, role, authorization identifier, or server location name	The thread is queued and suspended until another server thread becomes available and the number of concurrent active threads falls below the profile exception threshold. In ACTIVE mode, another thread becomes available when it deallocates. In INACTIVE mode, a thread becomes available when it becomes inactive. The connection that is associated with the thread remains open and is not terminated. The DB2 server suspends only as many threads as the profile exception threshold. When the total number of the threads being queued and suspended exceeds the profile exception threshold value for a particular profile ID, the DB2 server fails subsequent connection requests and returns SQLCODE -30041 to the client.

Table 20. Actions taken when thresholds for EXCEPTION profiles are exceeded (continued)

Filtering scope	Action
Collection identifier, package name, client user name, client application name, or client workstation name	The thread is queued and suspended until another server thread becomes available (that is, when another thread deallocates (in ACTIVE mode) or goes inactive (in INACTIVE mode)) and the number of concurrent active threads goes below the profile exception threshold. The connection that is associated with the thread remains open and is not terminated. The profile exception threshold value (ATTRIBUTE2) determines the maximum number of threads that are queued and suspended, as well as the maximum number of threads that can run concurrently. When the total number of threads that are being queued and suspended exceeds the ATTRIBUTE2 value for a particular profile ID, DB2 fails the subsequent SQL statement and returns SQLCODE -30041 to the client. For example, suppose that a profile for a package is started. That profile has an ATTRIBUTE2 value of 2. If five threads are started that request to run the package, two threads run concurrently, and two threads are queued and suspended. DB2 fails the SQL statement for the fifth thread.

The second part of the value in the ATTRIBUTE1 column controls the amount of detail in the message that DB2 issues when the threshold is met:

***type*_DIAGLEVEL1**
Where *type* is WARNING or EXCEPTION, DB2 takes the appropriate action for the specified type and issues a DSNT771I console message, which contains minimal information.

***type*_DIAGLEVEL2**
Where *type* is WARNING or EXCEPTION, DB2 takes the appropriate action for the specified type. DB2 also issues a DSNT772I console message, which contains additional information such as the profile ID and reason code in the message text.

The following table summarizes the meanings of the different columns of the DSN_PROFILE_ATTRIBUTES table for the MONITOR THREADS function.

Table 21. Summary of DSN_PROFILE_ATTRIBUTES values for the MONITOR THREADS function

KEYWORD	ATTRIBUTE1 column	ATTRIBUTE2 column	ATTRIBUTE3 column
MONITOR THREADS	Specify one of the following types of monitoring: • WARNING • WARNING_DIAGLEVEL1 • WARNING_DIAGLEVEL2 • EXCEPTION • EXCEPTION_DIAGLEVEL1 • EXCEPTION DIAGLEVEL2	Insert a value to indicate the threshold for the maximum allowed number of server threads that meet the profile criteria. The value that you specify must be less than or equal to the value of the MAXDBAT subsystem parameter.	Not used.

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related reference:

SYSIBM.DSN_PROFILE_TABLE

SYSIBM.DSN_PROFILE_ATTRIBUTES

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

Related information:

DSNT771I (DB2 Messages)

DSNT772I (DB2 Messages)

00E30505 (DB2 Codes)

00E30506 (DB2 Codes)

00E30507 (DB2 Codes)

00E30508 (DB2 Codes)

Monitoring idle threads by using profiles

You can monitor idle threads from remote connections.

Before you begin

Create a complete set of profile tables on the DB2 subsystem.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

Start an accounting trace that generates IFCID 0003 trace records, such as accounting class 1.

About this task

You can create warning and exception type MONITOR IDLE THREADS profiles. A *warning type* profile issues messages for threads that remain idle beyond the specified threshold, but it never cancels any threads. You can use the messages from warning type profiles to analyze the use of system resources by particular clients, applications, and users. You can also create *exception type* profiles that prioritize resources accordingly. An *exception type* profile issues messages and cancels threads when the specified threshold is reached.

The IDTHTOIN subsystem parameter never applies to any threads that meet the criteria for any MONITOR IDLE THREADS profile. Therefore, you can also use MONITOR IDLE THREADS to enable longer idle wait times for certain threads, without increasing the system-wide limit for idle thread timeouts. A zero value for either type of profile means that matching threads are allowed to remain idle indefinitely.

Important: Creating an exception type profile in conjunction with each warning type profile is recommended. Otherwise, any monitored threads can remain idle indefinitely.

Procedure

To monitor idle threads:

1. Insert values in the SYSIBM.DSN_PROFILE_TABLE table to create the profile and specify the scope of the profile:
   a. Insert a value in the PROFILEID column to identify the profile.
   b. Insert values to specify the filtering criteria: You can specify filtering criteria from any one of the following combinations:
      1) IP address or domain name, in the LOCATION column.
      2) Product identifier, in the PRDID column.
      3) Role and authorization identifier, in both ROLE and AUTHID columns.
      4) Role, in the ROLE column only.
      5) Authorization identifier, in the AUTHID column only.
      6) Server location name, location alias, or database name, in the LOCATION column.
      7) The location name of a requester, for monitored threads from a DB2 for z/OS requester.
      8) Collection identifier and package name, in both COLLID and PKGNAME columns.
      9) Collection identifier, in the COLLID column only.
      10) Package name, in the PKGNAME column only.
      11) Client application name, in the CLIENT_APPLNAME column.
      12) Client user identifier, in the CLIENT_USERID column.
      13) Client workstation name, in the CLIENT_WRKSTNNAME column.

      Other combinations of criteria are not accepted.You can specify several profiles that have overlapping criteria. DB2 might apply such overlapping profiles in combination, or it might apply one profile and exclude others, depending on the overlapping criteria.
2. Insert values in the SYSIBM.DSN_PROFILE_ATTRIBUTES table to specify the monitoring function of the profile:
   a. Insert a value in the PROFILEID column to identify which profiles use this type monitoring function. Insert a value that matches value of the PROFILEID column in the corresponding SYSIBM.DSN_PROFILE_TABLE rows.
   b. Specify the monitoring function by inserting the MONITOR IDLE THREADS value into the KEYWORD column in the DSN_PROFILE_ATTRIBUTES table. The profile monitors the approximate time (in seconds) that an active server thread is allowed to remain idle.
   c. Insert values in the ATTRIBUTE1 column of DSN_PROFILE_ATTRIBUTES table to specify how DB2 responds when a threshold is met. The possible values in the ATTRIBUTE1 column have two parts. The first part is the type of monitoring:

      WARNING
      A console message is issued at most every 5 minutes depending on the specified diagnosis level. When WARNING is specified, WARNING_DIAGLEVEL1 is used. Monitored threads are never

canceled, regardless of the IDTHTOIN subsystem parameter, unless a corresponding EXCEPTION profile is also created.

EXCEPTION
DB2 issues the messages that the diagnosis level specifies, and it cancels any active threads that remain idle as long as the specified threshold.

The second part of the value in the ATTRIBUTE1 column controls the amount of detail in the message that DB2 issues when the threshold is met:

***type*_DIAGLEVEL1**
Where *type* is WARNING or EXCEPTION, DB2 takes the appropriate action for the specified type and issues a DSNT771I console message, which contains minimal information.

***type*_DIAGLEVEL2**
Where *type* is WARNING or EXCEPTION, DB2 takes the appropriate action for the specified type. DB2 also issues a DSNT772I console message, which contains additional information such as the profile ID and reason code in the message text.

***type*_MESSAGE_FOR_IDLE_TIMEOUT**
type is WARNING. DB2 issues message DSNT773I to display information about the thread that exceeded the warning threshold. DB2 issues the DSNT773I message once for the thread while the thread remains in an idle state. After a COMMIT or ROLLBACK is performed, and no resources are active past the end of the unit of work, DB2 removes the warning against the thread. DB2 also issues console message DSNT771I to indicate how many times the timeout occurred.

d. Insert values in the ATTRIBUTE2 column of DSN_PROFILE_ATTRIBUTES table to specify the threshold that applies to active idle threads. You can specify any value that is valid for the IDTHTOIN subsystem parameter.

The following table summarizes the meanings of the different columns of the DSN_PROFILE_ATTRIBUTES table for the MONITOR IDLE THREADS function.

Table 22. Summary of DSN_PROFILE_ATTRIBUTES values for monitoring idle threads

KEYWORD	ATTRIBUTE1 column	ATTRIBUTE2 column	ATTRIBUTE3 column
MONITOR IDLE THREADS	Specify one of the following types of monitoring: • WARNING • WARNING_DIAGLEVEL1 • WARNING_DIAGLEVEL2 • EXCEPTION • EXCEPTION_DIAGLEVEL1 • EXCEPTION_DIAGLEVEL2	Insert a value to indicate the maximum number of seconds that active server threads are allowed to remain idle when they meet the profile criteria.	Not used.

3. Issue the START PROFILE command to start the monitoring functions that are specified in the profile tables.

What to do next

PSPI

Examine the accounting trace data that you obtained to determine which connection or thread exceeded a warning or exception level that was set by a monitor profile. The following trace fields provide that information:

QWAC_PROFMON_TYPE
Contains 'E' for an exception or a 'W' for a warning condition. Any other value indicates that a warning or exception did not occur.

QWAC_PROFMON_PID
Contains the profile ID of the monitor profile for the connection or thread that experienced the condition. The profile ID is the PROFILEID value in the SYSADM.DSN_PROFILE_TABLE. This value is a four-byte, binary integer.

PSPI

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related reference:

IDLE THREAD TIMEOUT field (IDTHTOIN subsystem parameter) (DB2 Installation and Migration)

-START PROFILE (DB2) (DB2 Commands)

SYSIBM.DSN_PROFILE_TABLE

SYSIBM.DSN_PROFILE_ATTRIBUTES

Related information:

DSNT771I (DB2 Messages)

DSNT772I (DB2 Messages)

DSNT773I (DB2 Messages)

00E30501 (DB2 Codes)

00E30502 (DB2 Codes)

Interactions between profiles for monitoring threads and connections

When multiple profiles specify overlapping criteria, DB2 might apply such profiles in combination. However, DB2 might also apply one profile and exclude others, depending on the overlapping criteria.

Filtering scopes for monitoring threads and connections

The filtering scopes for profiles that monitor system resources are organized into several categories. Whenever more than one profile specifies criteria from the same category and from the same monitor function, only one profile is applied.

Table 23. Categories and columns used to specify valid profiles for monitoring system resources

Filtering category	Columns to specify
Client IP address or domain name	Specify only the LOCATION column. The value can be an IP address or domain name. This category is the only accepted filtering criteria for profiles that specify the MONITOR CONNECTIONS.
Client product identifier	Specify only the PRDID column.
Role or authorization identifier	Specify one or all of the following columns: • ROLE • AUTHID Profiles that specify both ROLE and AUTHID take precedence over profiles that specify only one value. Profiles that specify only ROLE take precedence over profiles that specify only AUTHID
Collection identifier or package name	Specify only one or all of the following columns: • COLLID • PKGNAME Profiles that specify both COLLID and PKGNAME take precedence over profiles that specify only one value. Profiles that specify only COLLID take precedence over profiles that specify only PKGNAME
Location name, or location alias	Specify only the location name or location alias in LOCATION column. This category applies only to profiles that specify MONITOR THREADS and MONITOR IDLE THREADS.
Client application name, user ID, or workstation ID.	Specify only one of the following columns: • CLIENT_APPLNAME • CLIENT_USERID • CLIENT_WRKSTNNAME

Precedence of profiles for threads and connections

More than one profile might specify criteria that match the attributes of the same a thread or connection. DB2 applies only one profile when this happens for two profiles that specify the same monitoring function. The monitoring function is specified by the value in the matching row in the DSN_PROFILE_ATTRIBUTES table. The following rules apply to profiles that specify the same monitoring function and overlapping filtering criteria:

- When multiple criteria are specified in the same category in different matching profiles, DB2 uses only the profile that defines the criterion that takes precedence in that category:
  - ROLE takes precedence over AUTHID.
  - COLLID takes precedence over PKGNAME.
- When separate matching profiles specify different numbers of criteria from the same category, the more specifically defined profile is used. For example, when one profile specifies both ROLE and AUTHID, and another specifies only ROLE, DB2 uses only the profile that specifies both criteria.

- When criteria from different categories are specified in separate profiles, DB2 uses all of the profiles in combination.
- When a profile is defined that uses asterisk (*) values for filter criteria and other profiles define specific values, DB2 uses only the profile that specifies the specific values.
- If two or more profiles from the same category specify the same exactly matching filtering criteria, only the profile with the newer timestamp is applied.

The result of these rules is that only the most specifically matching profile from the same category, for any single monitoring function, is applied to any incoming thread or connection. However, every accepted thread or connection still counts toward the evaluation of the profile thresholds for subsequent incoming threads or connections. When two profiles specify separate monitoring functions, DB2 applies both profiles to the incoming threads or connections that qualify, regardless of whether the profile criteria are from the same filtering category.

Order of profile evaluation for threads and connections

When more than one profile applies to a thread or connection, the evaluation of the different profiles is not simultaneous. Instead the profiles are evaluated in the following order, according to the criteria that are specified in the profile:

1. IP address or domain name, in the LOCATION column.
2. Product identifier, in the PRDID column.
3. Role and authorization identifier, in both ROLE and AUTHID columns.
4. Role, in the ROLE column only.
5. Authorization identifier, in the AUTHID column only.
6. Server location name, location alias, or database name, in the LOCATION column.
7. The location name of a requester, for monitored threads from a DB2 for z/OS requester.
8. Collection identifier and package name, in both COLLID and PKGNAME columns.
9. Collection identifier, in the COLLID column only.
10. Package name, in the PKGNAME column only.
11. Client application name, in the CLIENT_APPLNAME column.
12. Client user identifier, in the CLIENT_USERID column.
13. Client workstation name, in the CLIENT_WRKSTNNAME column.

Only the first evaluated applicable profile is applied. Because the evaluation of multiple profiles is not simultaneous, the number of connections or threads on the subsystem might change during the evaluation of multiple profiles. Any profile that specifies a specific value in a particular column has precedence over a profile that specifies a single-byte asterisk value ('*') in the same column.

Related concepts:

Example profiles that monitor threads and connections

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

“Using profiles to monitor and optimize DB2 for z/OS subsystems” on page 657

Related reference:

Profile tables

Example profiles that monitor threads and connections

Examples are useful for helping you to understand the interactions between profiles that monitor system resources such as threads and connections.

The following example shows how DB2 determines which profiles to apply when the criteria of more than one profile match the attributes of a thread or connection. For example, assume that DSN_PROFILE_TABLE contains rows that contain the following values (some columns are not shown).

Table 24. Sample profiles in DSN_PROFILE_TABLE

PROFILEID	LOCATION	ROLE	AUTHID	PRDID	COLLID	PKGNAME
11	null	ROLE_DBA	null	null	null	null
12	null	null	USER1	null	null	null
13	null	ROLE_DBA	USER1	null	null	null
14	null	ROLE_APP	null	null	null	null
15	TEST.SVL. IBM.COM	null	null	null	null	null
16	null	null	null	SQL09073	null	null
17[1]	null	null	null	SQL09073	COLL1	null

Notes:

[1] The profile that is identified by PROFILEID=17 specifies values for columns in different filtering categories. Consequently, DB2 rejects this row when you issue the START PROFILE command.

The following examples assume that the DSN_PROFILE_ATTRIBUTES table contains rows that have matching values in the PROFILEID column, and that these rows also contain the same values in the KEYWORD column. Two profiles from the same category that specify different monitoring functions are both applied.

Consider example threads that have the following attributes:

ROLE='ROLE_APP' and AUTHID='USER1':
The criteria of profile 12 and profile 14 match the thread, but DB2 uses only profile 14 to evaluate whether to apply a threshold to the thread because ROLE takes precedence over AUTHID.

ROLE='ROLE_DBA' and AUTHID='USER2':
DB2 applies only the profile that is identified by PROFILEID=11.

ROLE='ROLE_DBA' and AUTHID='USER1':
The criteria of the following profiles match the thread: PROFILEID=11, PROFILEID=12, and PROFILEID=13. However DB2 applies only PROFILEID=13 to evaluate whether to apply a threshold against the thread. The profile that defines both ROLE and AUTHID takes precedence over a profile that defines only one of those values.

| In practice this result means that a profile that sets a lower threshold might be overruled by profile that specifies a greater threshold. For example, assume that the DSN_PROFILE_ATTRIBUTES table contains the rows shown in the following table.

Table 25. Sample rows in SYSIBM.DSN_PROFILE_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3
11	MONITOR THREADS	EXCEPTION	100	null
12	MONITOR THREADS	EXCEPTION	20	null
13	MONITOR THREADS	EXCEPTION	50	null

When you consider these values in combination with the values in combination with the values in Table 24 on page 121, you see that the following thresholds are created:

- “Profile 11” indicates that database administrators are accepted as many as 100 threads.
- “Profile 12” indicates that as many as 20 threads are accepted from the USER1 authorization ID.
- “Profile 13” indicates that as many as 50 threads are accepted for threads from the USER1 authorization ID under the ROLE_DBA role.

All of the example profiles specify the MONITOR THREADS function, and they all specify filtering criteria from the same category. So, only one of profiles applies to any particular thread. In this example, profile 13 applies to any thread that matches the AUTHID='USER1' and ROLE='ROLE_DBA' values. Therefore, because profile 13 takes precedence, profile 12 is never applied to any thread that meets both of these criteria. So, as many as 50 threads might be accepted from the 'USER1' authorization ID, before any action is taken.

However, profile 12 applies to any thread from 'USER1' under a different ROLE value, and every thread that has been accepted from 'USER1' (including any that also specified ROLE='ROLE_DBA') is now counted toward the evaluation of profile 12 in that case.

LOCATION='TEST.SVL.IBM.COM', ROLE='ROLE_APP', and PRDID='SQL09073':

The criteria of the following profiles match the thread: PROFILEID=14, PROFILEID=15, PROFILEID=16.

Because the criteria of these profiles are from separate filtering categories. So, DB2 uses all three profiles in combination to evaluate whether to apply thresholds to the thread. All of these profiles are applied, regardless of the functions or thresholds that are specified in the associated rows (which are not shown here) in the DSN_PROFILE_ATTRIBUTES table.

The following table shows partial sample data for certain columns in the SYSIBM.DSN_PROFILE_ATTRIBUTES table that specify how DB2 monitors threads and remote connections.

Table 26. Sample rows in SYSIBM.DSN_PROFILE_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE_TIMESTAMP
21	MONITOR THREADS	EXCEPTION	100	null	2008-12-19...
22	MONITOR IDLE THREADS	EXCEPTION	30	null	2008-12-17...

Table 26. Sample rows in SYSIBM.DSN_PROFILE_ATTRIBUTES (continued)

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE_ TIMESTAMP
23	MONITOR CONNECTIONS	WARNING	50	null	2009-01-21...

- "Profile 21" indicates that DB2 monitors active threads that meet the criteria defined by the DSN_PROFILE_TABLE row that contains 21 in the PROFILEID column. When the number of active threads exceeds 100, DB2 issues a message and suspends any new thread requests. When the number of the suspended threads exceeds 100, DB2 starts to reject any new thread request and issues SQLCODE -30041.
- "Profile 22" indicates that DB2 monitors idle threads that meet the criteria defined by the DSN_PROFILE_TABLE that contains 22 in the PROFILEID column. When a thread remains idle for more than 30 seconds, DB2 issues a message and terminates the idle thread.
- "Profile 23" indicates that DB2 monitors remote connections that meet the criteria defined the DSN_PROFILE_TABLE row that contains 23 in the PROFILEID column. When the number of remote connections reaches 50, DB2 issues a message and continues to service new connection requests.

Related concepts:

Interactions between profiles for monitoring threads and connections

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

"Using profiles to monitor and optimize DB2 for z/OS subsystems" on page 657

Related reference:

Profile tables

Variations on thread management

Transaction flow can vary in different environments and when dynamic SQL statements are executed.

TSO and call attachment facility

You can use the TSO attachment facility and call attachment facility (CAF) to request that SQL statements be executed in TSO foreground and batch. The processes differ from CICS or IMS transactions in that:

- No sign-on is required. The user is identified when the TSO address space is connected.
- Commit requires only a single-phase and only one I/O operation to each log. Single phase commit records are IFCID 0088 and 0089.
- Threads cannot be reused because the thread is allocated to the user address space.

Resource Recovery Services attachment facility (RRSAF)

With RRSAF, you have sign-on capabilities, the ability to reuse threads, and the ability to coordinate commit processing across different resource managers.

SQL under QMF

QMF uses CAF to create a thread when a request for work, such as a SELECT statement, is issued. A thread is maintained until the end of the session only if the requester and the server reside in different DB2 subsystems. If the requester and the server are both in the local DB2 subsystem, the thread is not maintained.

Related tasks:

Controlling connections (DB2 Administration Guide)

Monitoring and displaying RRSAF connections (DB2 Administration Guide)

Related information:

Managing QMF for TSO/CICS

Setting CICS options for threads

The CICS attachment facility provides a multithread connection to DB2 to allow you to operate DB2 with CICS

About this task

. Threads allow each CICS application transaction or DB2 command to access DB2 resources.

Procedure

Use the CICS resource definition online (RDO) to tune the CICS attachment facility and define the characteristics of your threads.
When a transaction needs a thread, an existing thread can be reused or a new thread can be created. If no existing thread is available, and if the maximum number of threads has not been reached, a thread is created.

Related concepts:

CICS Transaction Server for z/OS DB2 Guide

Related information:

DB2CONN resource definitions (CICS Transaction Server for z/OS)

SET DB2CONN (CICS Transaction Server for z/OS)

Overview of the CICS DB2 interface (CICS DB2 Guide)

Setting IMS options for threads

The IMS attachment facility provides a number of design options for threads.

Procedure

You can use the following IMS options for threads:

- Control the number of IMS regions connected to DB2. For IMS, this is also the maximum number of concurrent threads.
- A dependent region with a subsystem member (SSM) that is not empty is connected to DB2 at start up time. Regions with a null SSM cannot create a thread to DB2. A thread to DB2 is created at the first execution of an SQL statement in an IMS application schedule; it is terminated when the application terminates.

  The maximum number of concurrent threads used by IMS can be controlled by the number of IMS regions that can connect to DB2 by transaction class assignments. You can control the number by doing the following:
  - Minimize the number of regions needing a thread by the way in which you assign applications to regions.
  - Provide an empty SSM member for regions that does not connect to DB2.
- Optimize the number of concurrent threads used by IMS.
- Provide efficient thread reuse for high volume transactions.

  Thread creation and termination is a significant cost in IMS transactions. IMS transactions identified as wait for input (WFI) can reuse threads: they create a thread at the first execution of an SQL statement and reuse it until the region is terminated. In general, though, use WFI only for transactions that reach a region utilization of at least 75%.

  Some degree of thread reuse can also be achieved with IMS class scheduling, queuing, and a PROCLIM count greater than one. IMS Fast Path (IFP) dependent regions always reuse the DB2 thread.

Setting TSO options for threads

You can specify limits for the number of threads taken by the TSO and batch environments

Procedure

To tune your TSO attachment facility:

- Specify values for the following subsystem parameters:

  IDFORE
  The maximum number of TSO foreground connections (including DB2I, QMF, and foreground applications)

  IDBACK
  The maximum number of TSO background connections (including batch jobs and utilities)
- Because DB2 must be stopped to set new values, consider setting a higher IDBACK value for batch periods. The statistics record (IFCID 0001) provides information on the create thread queue. The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report shows (as shown in the example below) that information under the SUBSYSTEM SERVICES section.

Example

For TSO or batch environments, having 1% of the requests queued is probably a good number to aim for by adjusting the value of the CTHREAD subsystem parameter. Queuing at create thread time is not desirable in the CICS and IMS environments. If you are running IMS or CICS in the same DB2 subsystem as TSO and batch, use the values of the IDBACK and IDFORE subsystem parameters to limit the number of threads taken by the TSO and batch environments. The goal is to allow enough threads for CICS and IMS so that their threads do not queue. To determine the number of allied threads queued, see the QUEUED AT CREATE THREAD field (A) of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report.

```
SUBSYSTEM SERVICES          QUANTITY
---------------------------  --------
IDENTIFY                    30757.00
CREATE THREAD               30889.00
```

```
SIGNON                              0.00
TERMINATE                       61661.00
ROLLBACK                          644.00

COMMIT PHASE 1                      0.00
COMMIT PHASE 2                      0.00
READ ONLY COMMIT                    0.00

UNITS OF RECOVERY INDOUBT           0.00
UNITS OF REC.INDBT RESOLVED         0.00

SYNCHS(SINGLE PHASE COMMIT)     30265.00
QUEUED AT CREATE THREAD  A          0.00
SUBSYSTEM ALLIED MEMORY EOT         1.00
SUBSYSTEM ALLIED MEMORY EOM         0.00
SYSTEM EVENT CHECKPOINT             0.00
```

Figure 4. Thread queuing in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report

Related reference:

MAX TSO CONNECT field (IDFORE subsystem parameter) (DB2 Installation and Migration)

MAX BATCH CONNECT field (IDBACK subsystem parameter) (DB2 Installation and Migration)

MAX USERS field (CTHREAD subsystem parameter) (DB2 Installation and Migration)

Statistics Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Setting QMF options for threads

You can change the impact that QMF has on the performance of DB2 by specifying certain options in QMF.

Procedure

To set QMF performance options:

Specify the following options:
- The DSQSIROW parameter of the ISPSTART command
- SPACE parameter of the user QMF profile (Q.PROFILES)
- QMF region size and the spill file attributes
- TRACE parameter of the user QMF profile (Q.PROFILES)

Related information:

Managing QMF for TSO/CICS

Setting performance objectives for distributed workloads by using z/OS Workload Manager

You can use z/OS Workload Manager (WLM) support, to establish z/OS performance objectives for individual DDF server threads.

About this task

z/OS supports enclave system request blocks (SRBs). A z/OS enclave lets each thread have its own performance objective.

The z/OS performance objective of the DDF address space does not govern the performance objective of the user thread.

Procedure

Assign the DDF address space to a z/OS performance objective that is similar to the DB2 database services address space (*ssnm*DBM1). The z/OS performance objective of the DDF address space determines how quickly DB2 is able to perform operations associated with managing the distributed DB2 workload, such as adding new users or removing users that have terminated their connections. This performance objective should be a service class with a single velocity goal. This performance objective is assigned by modifying the WLM Classification Rules for started tasks (STC).

Related concepts:

z/OS performance options for DB2

Related tasks:

Determining z/OS Workload Manager velocity goals

Related reference:

MVS Planning: Workload Management (MVS Planning: Workload Management)

Classifying DDF threads

You can classify DDF threads by, among other things, authorization ID and stored procedure name. The stored procedure name is only used as a classification if the first statement issued by the client to being a new unit-of-work is an SQL CALL statement.

Procedure

Use the WLM administrative application to define the service classes you want z/OS to manage. These service classes are associated with performance objectives. When a WLM-established stored procedure call originates locally, it inherits the performance objective of the caller, such as TSO or CICS.

Important: If classification rules do not exist to classify some or all of your DDF transactions into service classes, those unclassified transactions are assigned to the default service class, SYSOTHER, which has no performance goal and is even lower in importance than a service class with a discretionary goal.

Classification attributes

Each of the WLM classification attributes has a two or three character abbreviation that you can use when entering the attribute on the WLM menus.

The following WLM classification attributes pertain to DB2 DDF threads:

AI Accounting information. The value of the DB2 accounting string associated with the DDF server thread, described by QMDAAINF in the DSNDQMDA mapping macro. WLM imposes a maximum length of 143 bytes for accounting information.

CI The DB2 correlation ID of the DDF server thread, described by QWHCCV in the DSNDQWHC mapping macro.

CN The DB2 collection name of the *first* SQL package accessed by the DRDA requester in the unit of work.

LU The VTAM LUNAME of the system that issued the SQL request.

NET The VTAM NETID of the system that issued the SQL request.

PC

Process name. This attribute can be used to classify the application name or the transaction name. The value is defined by QWHCEUTX in the DSNDQWHC mapping macro.

PK The name of the **first** DB2 package accessed by the DRDA requester in the unit of work.

PN The DB2 plan name of the requesting application.

PR Stored procedure name. This classification only applies if the first SQL statement from the client is a CALL statement.

SI Subsystem instance. The DB2 server's z/OS subsystem name.

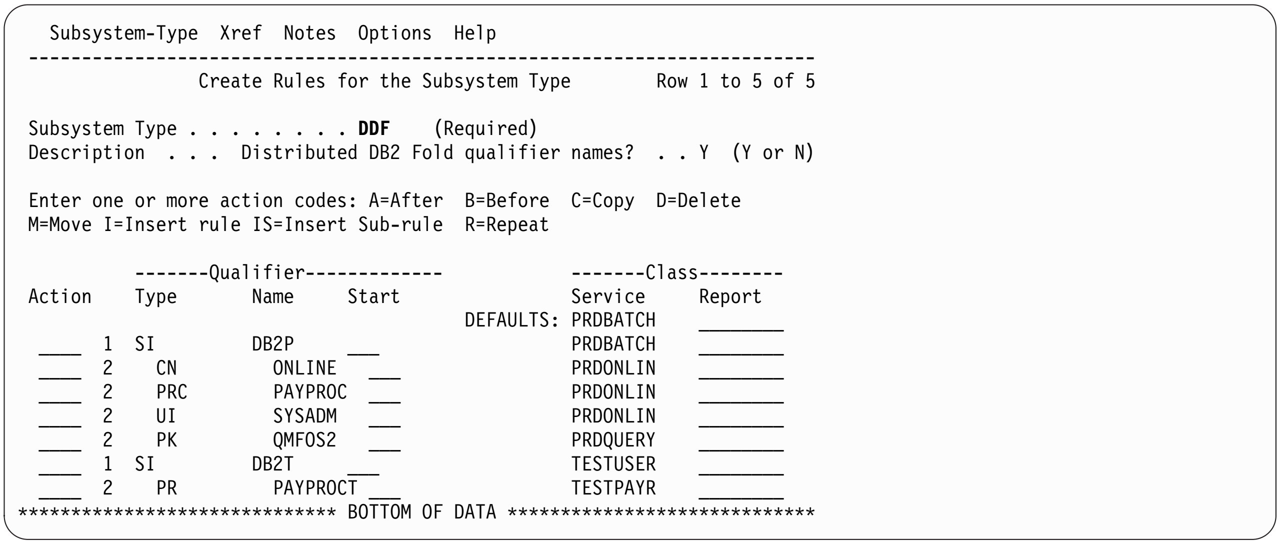
SPM Subsystem parameter. This qualifier has a maximum length of 255 bytes. The first 16 bytes contain the client's user ID. The next 18 bytes contain the client's workstation name. The remaining 221 bytes are reserved.

Important: If the length of the client's user ID is less than 16 bytes, uses blanks after the user ID to pad the length. If the length of the client's workstation name is less than 18 bytes, uses blanks after the workstation name to pad the length.

SSC Subsystem collection name. When the DB2 subsystem is a member of a DB2 data sharing group, this attribute can be used to classify the data sharing group name. The value is defined by QWHADSGN in the DSNDQWHA mapping macro.

UI User ID. The DDF server thread's primary authorization ID, after inbound name translation, which occurs only with SNA DRDA connections.

Figure 5 on page 129 shows how you can associate DDF threads with service classes.



```
   Subsystem-Type  Xref  Notes  Options  Help
 ---------------------------------------------------------------------------
                 Create Rules for the Subsystem Type        Row 1 to 5 of 5

 Subsystem Type . . . . . . . . DDF    (Required)
 Description  . . .  Distributed DB2 Fold qualifier names?  . . Y  (Y or N)

 Enter one or more action codes: A=After  B=Before  C=Copy  D=Delete
 M=Move I=Insert rule IS=Insert Sub-rule  R=Repeat

           -------Qualifier-------------            -------Class--------
 Action    Type       Name     Start                Service     Report
                                          DEFAULTS: PRDBATCH    ________
  ____  1  SI         DB2P     ___                  PRDBATCH    ________
  ____  2    CN         ONLINE   ___                PRDONLIN    ________
  ____  2    PRC        PAYPROC  ___                PRDONLIN    ________
  ____  2    UI         SYSADM   ___                PRDONLIN    ________
  ____  2    PK         QMFOS2   ___                PRDQUERY    ________
  ____  1  SI         DB2T     ___                  TESTUSER    ________
  ____  2    PR         PAYPROCT ___                TESTPAYR    ________
****************************** BOTTOM OF DATA *****************************
```

Figure 5. Classifying DDF threads using z/OS Workload Manager. You assign performance goals to service classes using the services classes menu of WLM.

The preceding figure shows the following classifications above:

- All DB2P applications accessing their first SQL package in the collection ONLINE are in service class PRDONLIN.
- All DB2P applications that call stored procedure PAYPROC first are in service class PRDONLIN.
- All work performed by DB2P user SYSADM is in service class PRDONLIN.
- Users other than SYSADM that run the DB2P PACKAGE QMFOS2 are in the PRDQUERY class. (The QMFOS2 package is not in collection ONLINE.
- All other work on the production system is in service class PRBBATCH.
- All users of the test DB2 system are assigned to the TESTUSER class except for work that first calls stored procedure PAYPROCT, which is in service class TESTPAYR.

Establishing performance periods for DDF threads

You can establish performance periods for DDF threads, including threads that run in the WLM-established stored procedures address space.

About this task

By establishing multiple performance periods, you can cause the thread's performance objectives to change based upon the thread's processor consumption. Thus, a long-running unit of work can move down the priority order and let short-running transactions get in and out at a higher priority.

Procedure

To design performance strategies for these threads:

- Take into account the events that cause a DDF thread to reset its z/OS performance period.
- Use velocity goals and use a single-period service class for threads that are always active. Because threads that are always active do not terminate the enclave and thus do not reset the performance period to period 1, a long-running thread always ends up in the last performance period. Any new

business units of work that use that thread suffer the performance consequences. This makes performance periods unattractive for long-running threads.

Establishing performance objectives for DDF threads

Threads are assigned a service class by the classification rules in the active WLM service policy. Each service class period has a performance objective (goal), and WLM raises or lowers that period's access to system resources as needed to meet the specified goal.

About this task

For example, the goal might be “application APPL8 should run in less than 3 seconds of elapsed time 90% of the time”.

No matter what your service class goals are, a request to start an address space might time out, depending on the timeout value that is specified in the TIMEOUT VALUE field of installation DSNTIPX. If the timeout value is too small, you might need to increase it to account for a busy system.

Procedure

To establish performance objectives for DDF threads and the related address spaces:

1. Create a WLM service definition that assigns service classes to the DDF threads under subsystem type DDF and to the DDF address space under subsystem type STC.
2. Install the service definition using the WLM menus and activate a policy (VARY WLM,POLICY=*policy*).

Chapter 10. Tuning parallel processing

You can improve the use of parallel processing by ensuring the availability of buffer pools, minimizing logical and physical contention and other situations that might reduce the degree of parallelism. Many of the following recommendations also apply to Sysplex query parallelism.

Procedure

PSPI

To tune parallel processing, use any of the following approaches:

- Increase the availability of buffer pools. Depending on buffer pool availability, DB2 might reduce the degree of parallelism (see `RAN REDUCED` in the example accounting trace report) or revert to a sequential plan before executing the parallel group (`SEQ - NO BUF` in the example accounting trace report).
  1. Check the QW0221C section in IFCID 0221 to determine which buffer pools is short on storage.
  2. Use the ALTER BUFFERPOOL command to increase the values of the following parameters:
     – VPSEQT, the sequential steal threshold
     – VPPSEQT, the parallel sequential threshold
     – VPXPSEQT, the assisting parallel sequential threshold, used only for Sysplex query parallelism.
  3. If increasing the parallel threshold parameters does not improve the degree of parallelism, use the ALTER BUFFERPOOL command to increase the total buffer pool size (VPSIZE). Use information from the statistics trace and the following formula trace to determine the amount of buffer space you need:

     ```
     (QBSTJIS / QBSTPQF) × 32 = buffer increase value
     ```

     QBSTJIS is the total number of requested prefetch I/O streams that were denied because of a storage shortage in the buffer pool. (There is one I/O stream per parallel task.) QBSTPQF is the total number of times that DB2 could not allocate enough buffer pages to allow a parallel group to run to the planned degree. For example, assume that the value of QBSTJIS is 100,000 and QBSTPQF is 2500 and apply the formula:

     ```
     (100,000 / 2500) × 32 = 1280
     ```

     In this case, you would use the ALTER BUFFERPOOL to increase the current VPSIZE by 2560 buffers to alleviate the degree degradation problem. You can use the DISPLAY BUFFERPOOL command to see the current value of the VPSIZE parameter.
- Minimize logical contention. For example, in a nested-loop join, the inner table can be in a partitioned or nonpartitioned table space, but DB2 is more likely to use a parallel join operation when the outer table exists in a partitioned table space.
- Minimize physical contention by putting data partitions on separate physical devices and keeping the number of partitions smaller than the number of internal paths in the controller.

- Check for updatable cursors. At run time, DB2 might determine that an ambiguous cursor is updatable. This appears in the SEQ - CURSOR field in example accounting trace report
- Check for proper hardware and software support. If you do not have the hardware sort facility at run time, and a sort merge join is needed, you see a value in the SEQ - NO ESA field.
- Check whether the configuration of online processors has changed. If fewer online processors are available at run time than at bind time, DB2 reformulates the parallel degree to take best advantage of the current processing power. This reformulation is indicated by a value in theREFORM PARAL-CONFIG field in the accounting report.

  The following example, shows part of an accounting trace report.

```
TIMES/EVENTS  APPL (CLASS 1)   DB2 (CLASS 2)    CLASS 3 SUSP.   ELAPSED TIME
------------  --------------  --------------    --------------  ------------
ELAPSED TIME       32.578741       32.312218    LOCK/LATCH         25.461371
 NONNESTED         28.820003       30.225885    SYNCHRON. I/O       0.142382
 STORED PROC        3.758738        2.086333     DATABASE I/O       0.116320
 UDF                0.000000        0.000000     LOG WRTE I/O       0.026062
 TRIGGER            0.000000        0.000000    OTHER READ I/O   3:00.404769
                                                OTHER WRTE I/O      0.000000
CPU CP TIME      1:29.695300     1:29.644026    SER.TASK SWTCH      0.000000
 AGENT              0.225153        0.178128     UPDATE COMMIT      0.000000
  NONNESTED         0.132351        0.088834     OPEN/CLOSE         0.000000
  STORED PRC        0.092802        0.089294     SYSLGRNG REC       0.000000
  UDF               0.000000        0.000000     EXT/DEL/DEF        0.000000
  TRIGGER           0.000000        0.000000     OTHER SERVICE      0.000000
 PAR.TASKS       1:29.470147     1:29.465898    ARC.LOG(QUIES)      0.000000

⋮
...           QUERY PARALLEL.     TOTAL
              ---------------  --------
              MAXIMUM MEMBERS        1
              MAXIMUM DEGREE        10
              GROUPS EXECUTED        1
               RAN AS PLANNED        1
               RAN REDUCED           0
               ONE  COOR=N           0
               ONE  ISOLAT           0
               ONE  DCL TTABLE       0
               SEQ - CURSOR          0
               SEQ - NO ESA          0
               SEQ - NO BUF          0
               SEQ - ENCL.SER.       0
              MEMB SKIPPED(%)        0
              DISABLED BY RLF       NO
              REFORM PARAL-CONFIG    0
              REFORM PARAL-NO BUF    0
```

Figure 6. A partial sample that shows parallelism fields in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting trace report

- Set the value of the PARA_EFF subsystem parameter to control the parallelism efficiency that DB2 assumes when selecting access paths. The recommended setting is 50. The efficiency of parallelism depends on system configuration and workload.

  Perfect parallelism efficiency assumes that DB2 is able to distribute the work uniformly across all parallel tasks and that the system resources (processor, memory, bufferpool) exist to process the expected degree of parallelism. Often, the distribution of work is not uniform and the processing resources are not available to handle all of the parallel tasks concurrently. Consequently, assuming perfect parallelism efficiency for the purposes of deciding to switch to a more parallel plan from a less parallel plan can result in inefficient performance. The value of the PARA_EFF subsystem parameter has the following meanings:

100 DB2 uses the most-optimistic assumption regarding the cost reductions that result from parallelism efficiency. However, DB2 might overestimate the cost reduction that a higher degree of parallelism provides. When that happens DB2 might select an access path that achieves a higher degree of parallelism but actually yields neither the estimated cost savings nor a shorter elapsed time.

1-99 DB2 considers cost reductions that result from parallelism efficiency, but it uses a proportionally reduced estimation of the cost reduction that degrees of parallelism provide. The closer the value to 1, the less effect that the expected degree of parallelism has on the access path selection. Whereas, the closer that the value is to 100, the more likely DB2 is to choose an access path that provides a higher estimated degree of parallelism, even though the estimated or actual cost might be greater.

0 DB2 chooses the access path that has the lowest sequential cost, regardless any consideration of the estimated cost reduction by parallelism. However, this setting only removes the consideration of parallelism during access path selection. It does not prevent the use of parallelism for the selected access path.

PSPI

Related tasks:

Programming for parallel processing

Interpreting query parallelism

Enabling parallel processing

Related reference:

Buffer pool thresholds that you can change

PARALLELISM EFFICIENCY field (PARA_EFF subsystem parameter) (DB2 Installation and Migration)

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

Accounting Long Report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Disabling query parallelism

You can prevent DB2 from using parallel processing.

Procedure

To disable parallel operations, do any of the following actions:

- For static SQL, rebind the package and specify the DEGREE(1) bind option.
- GUPI

  For dynamic SQL, issue the following SQL statement:

```
SET CURRENT DEGREE = '1';
```

The default value for CURRENT DEGREE is 1 unless your installation has changed the default for the CURRENT DEGREE special register.

GUPI

- Set the parallel sequential threshold (VPPSEQT) to 0.
- Insert rows in a resource limit table (DSNRLST*xx*) to restrict the parallelism mode:
  1. Specify the RLFFUNC value for each type of parallelism that you want to disable:

     Query I/O parallelism
     Insert a row that contains the RLFFUNC='3' value. Query I/O parallelism is deprecated and is likely to be removed in a future release.

     CP parallelism
     Insert a row that contains the RLFFUNC='4' value.

     Sysplex query parallelism
     Insert a row that contains the RLFFUNC='5' value. Sysplex query parallelism is deprecated and is likely to be removed in a future release.

     To disable all query parallelism for a dynamic query, you must insert a separate row for each possible mode of parallelism.
  2. Qualify the rows according to the following rules: Qualifying by plan or by package are not separate functions for parallelism, as they are for predictive and reactive governing:
     - When the row specifies a plan name, DB2 finds the row only for queries that are executed from the plan.
     - When the row specifies a package name, DB2 finds the row only for queries that are executed from the package.

     The values of the RLFCOLLN, RLFPKG, and RLFPLAN columns can be blank for rows that are qualified by authorization ID only.

  If parallelism is disabled for a query, the query runs sequentially. If no entry can be found in your resource limit table that applies to parallelism, or if your resource limit table cannot be read, the resource limit facility does not disable query parallelism.

Results

Example

If the following resource limit table is active, it causes the following effects:

- Disables I/O parallelism for all dynamic queries in the IOHOG package.
- Disables CP parallelism and Sysplex query parallelism for all dynamic queries in the CPUHOG package.

Table 27. Example RLST to govern query parallelism

RLFFUNC	AUTHID	LUNAME	RLFCOLLN	RLFPKG
3	(blank)	PUBLIC	blank	IOHOG
4	(blank)	PUBLIC	blank	CPUHOG

Table 27. Example RLST to govern query parallelism (continued)

RLFFUNC	AUTHID	LUNAME	RLFCOLLN	RLFPKG
5	(blank)	PUBLIC	blank	CPUHOG

What to do next

PSPI

To determine whether parallelism has been disabled by a value in your resource limit specification table (DSNRLST*xx*), look for a non-zero value in field QXRLFDPA in IFCID 0002 or 0003. The QW0022RP field in IFCID 0022 indicates whether this particular statement was disabled.

PSPI

Related concepts:

Parallel processing

Related tasks:

Specifying and changing resource limits

Interpreting query parallelism

Tuning parallel processing

Enabling parallel processing

Related reference:

DEGREE bind option (DB2 Commands)

SET CURRENT DEGREE (DB2 SQL)

CURRENT DEGREE (DB2 SQL)

DSNRLSTxx resource limit tables

Buffer pool thresholds that you can change

Chapter 11. Improving the performance of stored procedures and user-defined functions

You can improve the performance of stored procedures and user-defined functions by following certain recommendations.

Procedure

To improve the performance of stored procedures and user-defined functions, use any of the following recommendations:

- Update the ASUTIME column of the SYSIBM.SYSROUTINES catalog table to set processor limits for each stored procedures or function. The limits that you specify enable DB2 to cancel procedures or functions that loop.
- Limit the number of times that a stored procedure can terminate abnormally by specifying one of the following options:
  - The MAX ABEND COUNT field on installation panel DSNTIPX. The limit that you specify applies to all stored procedures and prevents a problem procedure from overwhelming the system with abend dump processing.
  - The STOP AFTER FAILURES option on the ALTER or CREATE PROCEDURE statement. The limit that you specify overrides the system limit that is specified in the MAX ABEND COUNT field to specify limits for specific stored procedures.
- Maximize the number of procedures or functions that can run concurrently in a WLM-established stored procedure address space.
- Group your stored procedures in WLM application environments. For more information, see Defining application environments.
- Use indicator variables in your programs and pass the indicator variables as parameters. When output parameters occupy a large amount of storage, passing the entire storage areas to your stored procedure can be wasteful. However, you can use indicator variables in the calling program to pass only a two-byte area to the stored procedure and receive the entire area from the stored procedure.
- Set a high-enough priority for the WLM-managed stored procedures address spaces.
- Set the performance-related options appropriately in the CREATE PROCEDURE statement. The following table shows the recommended values.

Table 28. Recommended values for performance-related options in the CREATE procedure statement.

Option	Recommend setting
PROGRAM TYPE	SUB
STAY RESIDENT	YES
PARAMETER STYLE	GENERAL WITH NULLS or SQL
COMMIT ON RETURN	NO for stored procedures that are called locally; YES for stored procedures that are called from distributed client applications in environments where sysplex workload balancing is not used.

- Do not use the DSNTRACE DD statement in any of your stored procedures address space startup procedures. DSNTRACE is a facility that can be used to capture all trace messages for offline reference and diagnosis. However, DSNTRACE greatly increases the stored procedure initialization overhead. Also, DSNTRACE does not function in a multitasking environment because the CAF does not serialize access to the DSNTRACE trace data set.

- Specify a large enough value for the CACHERAC subsystem parameter on DSNTIP installation panel. The CACHERAC parameter specifies how much storage to allocate for the caching of routine authorization information for all routines on DB2 the member.
- Set the CMTSTAT subsystem parameter to INACTIVE This setting causes distributed threads to become inactive at commit when possible. The inactive threads become available for thread reuse, and that reduces the amount of thread storage needed for the workload, by reducing the number of distributed active threads.
- Convert external stored procedures to native SQL procedures whenever possible. The body of a native SQL procedure is written in SQL, and DB2 does not generate an associated C program for native stored procedures. Native procedures typically perform better and have more functionality that external procedures.
- Study your workload for external stored procedures and functions carefully. You can use DB2 Performance Expert of DB2 Performance Monitor to monitor stored procedures and user-defined functions.
- Use partitioned data set extended (PDSE) members for load libraries that contain stored procedures. By using PDSE members, you might eliminate the need to stop and start the stored procedures address space because of growth in load libraries, because the new extent information is dynamically updated. If a load library grows from additions or replacements, the library might have to be extended. If you use partitioned data set (PDS) members, load failures might occur because the new extent information is not available.

Related tasks:

Limiting resources for a stored procedure

Creating a native SQL procedure (DB2 Application programming and SQL)

Passing large output parameters to stored procedures by using indicator variables (DB2 Application programming and SQL)

Defining application environments (MVS Planning: Workload Management)

Related reference:

ROUTINE AUTH CACHE field (CACHERAC subsystem parameter) (DB2 Installation and Migration)

Case study: Stored procedure that runs RUNSTATS in parallel (DB2 for z/OS Stored Procedures: Through the CALL and Beyond)

Related information:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

Maximizing the number of procedures or functions that run in an address space

You can improve the performance of stored procedures and user-defined functions by maximizing the number of procedures or functions that can run concurrently in a WLM-established stored procedures address space.

About this task

Each task control block that runs in a WLM-established stored procedures address space uses approximately 200 KB below the 16-MB line. DB2 needs this storage for stored procedures and user-defined functions because you can create both main programs and subprograms, and DB2 must create an environment for each.

Procedure

To maximize the number of procedures or functions that can run concurrently in a WLM-established stored procedures address space:

- Set the region size for the address spaces to REGION=0 to obtain the largest possible amount of storage below the 16-MB line.
- Limit storage required by application programs below the 16-MB line by using the following methods:
  - Link edit programs above the line with AMODE(31) and RMODE(ANY) attributes
  - Use the RES and DATA(31) compiler options for COBOL programs
- Limit storage required by Language Environment® by using the following run time options:

  HEAP(,,ANY)
  Allocates program heap storage above the 16-MB line

  STACK(,,ANY,)
  Allocates program stack storage above the 16-MB line

  STORAGE(,,,4K)
  Reduces reserve storage area below the line to 4 KB

  BELOWHEAP(4K,,)
  Reduces the heap storage below the line to 4 KB

  LIBSTACK(4K,,)
  Reduces the library stack below the line to 4 KB

  ALL31(ON)
  Causes all programs contained in the external user-defined function to execute with AMODE(31) and RMODE(ANY)

  The definer can list these options as values of the RUN OPTIONS parameter of CREATE FUNCTION, or the system administrator can establish these options as defaults during Language Environment installation. For example, the RUN OPTIONS option parameter could contain:

  ```
  H(,,ANY),STAC(,,ANY,),STO(,,,4K),BE(4K,,),LIBS(4K,,),ALL31(ON)
  ```

- Set the NUMTCB parameter for WLM-established stored procedures address spaces to a value greater than 1 to allow more than one TCB run in an address space. Be aware that setting NUMTCB to a value greater than 1 also reduces your level of application program isolation. For example, a bad pointer in one application can overwrite memory that is allocated by another application.

  A stored procedure can invoke only one utility in one address space at any given time because of the resource requirements of utilities. On the WLM Application-Environment panel, set NUMTCB to 1.With NUMTCB=1 or the value of NUMTCB being forced to 1, multiple WLM address spaces are created to run each concurrent utility request that comes from a stored procedure call.

Related tasks:

Specifying the number of stored procedures that can run concurrently (DB2 Application programming and SQL)

Assigning stored procedures and functions to WLM application environments

You can assign procedures to WLM environments to route the work that is associated with the procedures to specific address spaces.

About this task

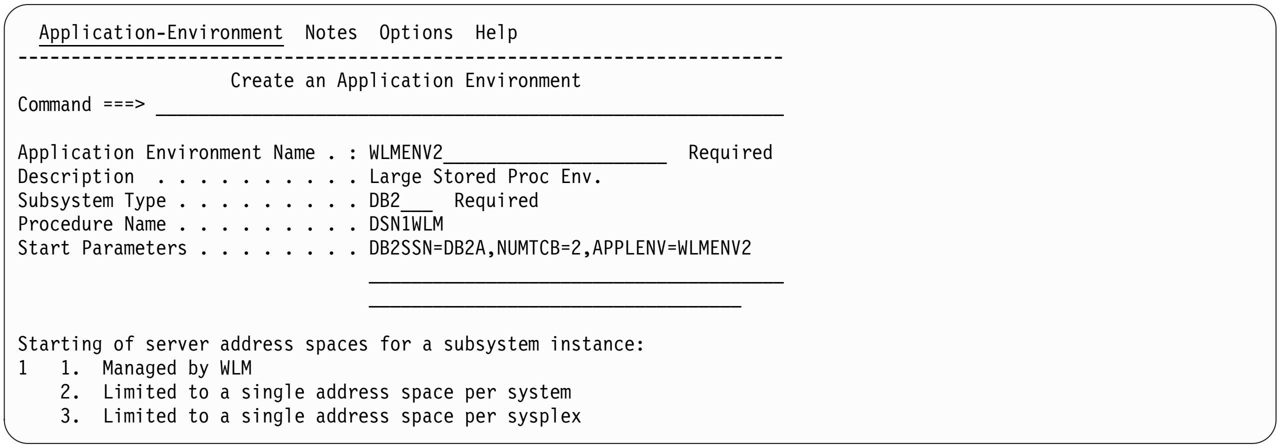
Workload manager routes work to address spaces based on the application environment name and service class associated with the stored procedure or function.

To assign a stored procedures or user-defined functions to run in WLM-established stored procedures address spaces:

Procedure

1. Make sure a numeric value is specified for the STORTIME subsystem parameter. If you have problems with setting up the environment, this timeout value ensures that your request to execute a stored procedure does not wait for an unlimited amount of time.

   To prevent creating too many address spaces, create a relatively small number of WLM application environments and z/OS service classes.
2. Minimize the number of application environments and z/OS service classes. Otherwise, you might cause WLM to create too many WLM address spaces. WLM creates one address space for each combination of application environment and service class. In other words, if five application environments have calling threads, and six service classes exist, WLM might create as many as 30 address spaces.
3. Use the WLM application environment panels to associate the environment name with the JCL procedure. The following figure shows an example of this panel. For detailed information about workload management panels and how to use them, see Using the WLM ISPF Application (MVS Planning: Workload Management).



```
  Application-Environment  Notes  Options  Help
------------------------------------------------------------------------
                    Create an Application Environment
Command ===> ___________________________________________________________

Application Environment Name . : WLMENV2____________________  Required
Description  . . . . . . . . . . Large Stored Proc Env.
Subsystem Type . . . . . . . . . DB2___  Required
Procedure Name . . . . . . . . . DSN1WLM
Start Parameters . . . . . . . . DB2SSN=DB2A,NUMTCB=2,APPLENV=WLMENV2
                                 _______________________________________
                                 ___________________________________

Starting of server address spaces for a subsystem instance:
1   1.  Managed by WLM
    2.  Limited to a single address space per system
    3.  Limited to a single address space per sysplex
```

Figure 7. WLM panel to create an application environment. You can also use the variable &IWMSSNM for the DB2SSN parameter (DB2SSN=&IWMSSNM). This variable represents the name of the subsystem for which you are starting this address space. This variable is useful for using the same JCL procedure for multiple DB2 subsystems.

4. Specify the WLM application environment name for the WLM_ENVIRONMENT option of the CREATE or ALTER PROCEDURE (or FUNCTION) statement to associate a stored procedure or user-defined function with an application environment.
5. Using the installation utility in the WLM application, install the WLM service definition that contains information about this application environment into the couple data set.
6. Activate a WLM policy from the installed service definition.
7. Begin running stored procedures.

Recommendations for assigning stored procedures to WLM environments

The NUMTCB value that you choose for each application environment will vary by language. The following table provides general recommendations for the WLM procedure NUMTCB setting for the different language stored procedures. These recommendations are based on available resources and should be tuned accordingly.

Table 29. Recommended types WLM environments to define

Stored procedure name	NUMTCB value	Comments
COBOL, C/C++, PL/I	10-40	
Debugging COBOL, C/C++ and PL/I related stored procedures	10-40	
REXX	1	REXX stored procedures must run in a WLM procedure with NUMTCB = 1. If they execute in a procedure with NUMTCB>1, unpredictable results, such as an 0C4 will occur.
Java™ (non-resettable mode)	20-40	Each JVM is started in non-resettable mode and is never reset. This allows you to run many more Java stored procedures.
External SQL stored procedures	10-40	Must have one unauthorized data set. COBOL, C/C++, and PL/I stored procedures can share the same application environment if the STEPLIB data sets and run time options are the same for all languages.

Related concepts:

WLM Application Environment recommendations (DB2 for z/OS Stored Procedures: Through the CALL and Beyond)

Related tasks:

Managing authorizations for creation of stored procedures in WLM environments (Managing Security)

Altering stored procedures (DB2 Administration Guide)

Specifying the number of stored procedures that can run concurrently (DB2 Application programming and SQL)

Setting up a WLM application environment for stored procedures during installation (DB2 Installation and Migration)

Defining application environments (MVS Planning: Workload Management)

Creating the required WLM application environment (DB2 for z/OS Stored Procedures: Through the CALL and Beyond)

Setting up WLM for DB2 stored procedures (DB2 for z/OS Stored Procedures: Through the CALL and Beyond)

Related reference:

TIMEOUT VALUE field (STORTIME subsystem parameter) (DB2 Installation and Migration)

CREATE PROCEDURE (DB2 SQL)

ALTER PROCEDURE (external) (DB2 SQL)

ALTER PROCEDURE (SQL - native) (DB2 SQL)

ALTER PROCEDURE (SQL - external) (DB2 SQL)

Accounting for nested activities

The accounting class 1 and class 2 CPU and elapsed times for triggers, stored procedures, and user-defined functions are accumulated in separate fields and exclude any time accumulated in other nested activity.

These CPU and elapsed times are accumulated for each category during the execution of each agent until agent deallocation. Package accounting can be used to break out accounting data for execution of individual stored procedures, user-defined functions, or triggers. The following figure shows an agent that executes multiple types of DB2 nested activities.

```
Time   Application   DB2                           Stored procedure            User-defined function
-----  ------------  ----------------------------  --------------------------  --------------------------
 T0    Code
 T1    SQL---------->
 T2       <---------
 T3    SQL---------->
 T4                  Trigger
 T5                  SQL
 T6                  CALL triggered--------------->
 T7                                                Stored procedure code
 T8                                 <-------------SQL
 T9                                 ------------->Stored procedure code
 T10                                <-------------SQL(User-defined function)
 T11                 Start User-defined function
 T12                 --------------------------------------------------------->User-defined function code
 T13                 <---------------------------------------------------------SQL
 T14                 --------------------------------------------------------->User-defined function code
 T16                 <---------------------------------------------------------User-defined function ends
 T17                 Back to Stored procedure----->Stored procedure code
 T18                 SQL             <-------------Back to trigger
 T19                 Trigger ends
 T20  Code<----------Return to Application
 T21  End
```

Figure 8. Time spent executing nested activities

The following table shows the formula used to determine time for nested activities.

Table 30. Sample for time used for execution of nested activities

Count for	Formula	Class
Application elapsed	T21-T0	1
Application task control block (TU)	T21-T0	1
Application in DB2 elapsed	T2-T1 + T4-T3 + T20-T19	2
Application in DB2 task control block (TU)	T2-T1 + T4-T3 + T20-T19	2
Trigger in DB2 elapsed	T6-T4 + T19-T18	2
Trigger in DB2 task control block (TU)	T6-T4 + T19-T18	2
Wait for STP time	T7-T6 + T18–T17	3
Stored procedure elapsed	T11-T6 + T18-T16	1
Stored procedure task control block (TU)	T11-T6 + T18-T16	1
Stored procedure SQL elapsed	T9-T8 + T11-T10 + T17-16	2
Stored procedure SQL elapsed	T9-T8 + T11-T10 + T17-T16	2
Wait for user-defined function time	T12-T11	3
User-defined function elapsed	T16-T11	1
User-defined function task control block (TU)	T16-T11	1
User-defined function SQL elapsed	T14-T13	2
User-defined function SQL task control block (TU)	T14-T13	2

Note: In the preceding table, TU = time used.

The total class 2 time is the total of the "in DB2" times for the application, trigger, stored procedure, and user-defined function. The class 1 "wait" times for the stored procedures and user-defined functions need to be added to the total class 3 times.

Providing cost information, for accessing user-defined table functions, to DB2

User-defined table functions add additional access cost to the execution of an SQL statement.

About this task

PSPI

For DB2 to factor in the effect of user-defined table functions in the selection of the best access path for an SQL statement, the total cost of the user-defined table function must be determined.

The total cost of a table function consists of the following three components:

- The initialization cost that results from the first call processing
- The cost that is associated with acquiring a single row
- The final call cost that performs the clean up processing

These costs, though, are not known to DB2 when I/O costs are added to the CPU cost.

Procedure

To assist DB2 in determining the cost of user-defined table functions:

Use the following fields in SYSIBM.SYSROUTINES catalog table:

IOS_PER_INVOC
The estimated number of I/Os per row.

INSTS_PER_INVOC
The estimated number of instructions.

INITIAL_IOS
The estimated number of I/Os performed the first and last time the function is invoked.

INITIAL_INSTS
The estimated number of instructions for the first and last time the function is invoked.

These values, along with the CARDINALITY value of the table being accessed, are used by DB2 to determine the cost. The results of the calculations can influence such things as the join sequence for a multi-table join and the cost estimates generated for and used in predictive governing.
You can determine values for the four fields by examining the source code for the table function:

1. Estimate the I/Os by examining the code executed during the FIRST call and FINAL call.
2. Look for the code executed during the OPEN, FETCH, and CLOSE calls.
3. The costs for the OPEN and CLOSE calls can be amortized over the expected number of rows returned.
4. Estimate the I/O cost by providing the number of I/Os that are issued. Include the I/Os for any file access.
5. Calculate the instruction cost by counting the number of high level instructions executed in the user-defined table function and multiplying it by a factor of 20. For assembler programs, the instruction cost is the number of assembler instructions.

Example

The following statement shows how these fields can be updated. The authority to update is the same authority as that required to update any catalog statistics column.

```
UPDATE SYSIBM.SYSROUTINES SET
   IOS_PER_INVOC   = 0.0,
   INSTS_PER_INVOC = 4.5E3,
   INITIAL_IOS     = 2.0
   INITIAL_INSTS   = 1.0E4,
   CARDINALITY     = 5E3
WHERE
   SCHEMA = 'SYSADM' AND
   SPECIFICNAME = 'FUNCTION1' AND
   ROUTINETYPE = 'F';
```

PSPI

Part 3. Controlling resource usage

When system resources are shared among transactions, user queries, web requests, distributed application requests, and batch programs, you need to control how those resources are used, separate data, and set priorities carefully.

About this task

You might choose to emphasize resource use, performance, concurrency, or data security. Many of the things you currently do to improve response time or reduce processor consumption for a single DB2 subsystem also apply in the data sharing environment.

Procedure

1. Choose the controls that best match your goals. For example, you might want to:
   - Minimize resource usage
   - Maximize throughput
   - Maximize response time
   - Ensure a certain level of service to some users
   - Avoid conflicts between users

   Consequently, Your goal might be to favor a certain class of users or to achieve the best overall system performance.
2. Use the appropriate facilities to tune the performance of your system:

Option	Description
Prioritizing resources	z/OS workload management (WLM) controls the execution of DB2 work based on the priorities that you set. In CICS environments without the Open Transaction Environment (OTE) function, DB2 work and application work is performed in different tasks. DB2 work is managed at the subtask level. With CICS OTE, DB2 work and application work can be performed in the same task. You can manage the DB2 subtasks through various settings in the CICS resource definition online (RDO). Without OTE, some overhead is incurred for each task switch. Therefore, depending on the SQL activity, CICS OTE can improve performance significantly because of the reduction of needing to switch tasks. In other environments such as batch and TSO, which typically have a single task requesting DB2 services, the task-level processor dispatching priority is irrelevant. Access to processor and I/O resources for synchronous portions of the request is governed solely by WLM. **Related information:** MVS Planning: Workload Management (MVS Planning: Workload Management) (WLM) z/OS MVS Initialization and Tuning Guide (WLM) The System Resources Manager (MVS Initialization and Tuning Guide) (WLM) Enabling CICS DB2 applications to use the open transaction environment (OTE) (CICS Transaction Server for z/OS) (CICS)
Limiting resources for each job	You use the TIME parameter of a job or step to control the total amount of processor resources used for a job, instead of the amount used by a single query. Because most of resource usage occurs within the standard job structure, you can control processor usage at the job level. **Related information:** Processing control by timing execution (MVS JCL User's Guide) (JCL)

Option	Description
Limiting resources for TSO sessions	You can control the amount of resources used for an entire TSO session. Time limits can apply to either TSO sessions or to batch jobs. Your z/OS system programmer can provide a time parameter on the logon procedure or on a job statement in the logon pre-prompt exit. This time limit applies to the session, rather than for an individual query or a single program. If you want to control the amount of resources used for an entire TSO session, rather than the amount used by a single query, use this control. **Related information:** Controlling TSO connections (DB2 Administration Guide) Customizing the logon and logoff process (TSO/E Customization) (TSO/E) Processing control by timing execution (MVS JCL User's Guide) (JCL)
Limiting resources for IMS and CICS	Various IMS and CICS controls (such as the PROCLIM keyword of the TRANSACT macro in IMS). **Related information:** Tuning the system (IMS) Avoiding contention for IMS resources (excluding buffer pools) (IMS) TRANSACT macro (IMS) Controlling IMS connections (DB2 Administration Guide) Improving performance (CICS Transaction Server for z/OS) (CICS) Search the CICS Library. (CICS) Controlling CICS connections (DB2 Administration Guide)
Limiting resources for stored procedures	Use the ASUTIME column of SYSIBM.SYSROUTINES catalog table and the MAX ABEND COUNT field on installation panel DSNTIPX. **Related information:** Limiting resources for a stored procedure SYSIBM.SYSROUTINES table (DB2 SQL) MAX ABEND COUNT field (STORMXAB subsystem parameter) (DB2 Installation and Migration)
Reducing locking contention	Use DB2 locking parameters, DISPLAY DATABASE LOCKS, lock trace data, database design **Related information:** Improving concurrency Designing databases for concurrency Monitoring concurrency and locks -DISPLAY DATABASE (DB2) (DB2 Commands)

Option	Description
Limiting the execution time of dynamic statements	Use the DB2 resource limit facility (governor). **Related information:** Resource limit facility controls Avoiding contention for IMS resources (excluding buffer pools)
Controlling use of parallelism	DB2 resource limit facility, SET CURRENT DEGREE statement **Related information:** Disabling query parallelism SET CURRENT DEGREE (DB2 SQL) CURRENT DEGREE (DB2 SQL)
Controlling the use of system resources by threads and connections	Use profile tables and certain DB2 subsystem parameters. **Related information:** Managing DB2 threads Monitoring threads and connections by using profiles Monitoring threads (DB2 Administration Guide) Profile tables Setting thread limits for database access threads
Evaluating long-term resource usage	Use accounting trace data, Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports. **Related information:** Accounting trace Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS
Predicting resource consumption	Use EXPLAIN table data, Visual Explain, and the predictive governing capability of the resource limit facility. **Related information:** Investigating SQL performance by using EXPLAIN EXPLAIN (DB2 SQL) Limiting resources for SQL statements predictively Setting limits for system resource usage by using the resource limit facility

Related concepts:

Performance monitoring and tuning for data sharing environments (DB2 Data Sharing Planning and Administration)

Related tasks:

Improving performance for applications that access distributed data

“Monitoring threads and connections by using profiles” on page 108

Chapter 12. The DB2 system monitor

The DB2 system monitor starts automatically and identifies problems related to CPU stalls and DBM1 below-the-bar storage.

The DB2 system monitor looks for CPU stalls that result in latch contention. When it detects a CPU stall, the DB2 system monitor attempts to clear the latch contention by temporarily boosting WLM priority. In addition, the system monitor issues the following messages:

- DSNV508I, to report when DBM1 storage that is below the 2-GB bar reaches critical storage thresholds, and information about the amounts of consumed and available storage.
- A series of DSNV512I messages to identify the agents that consume the most storage.

Specify aggressive performance goals in WLM and very high importance for the *ssnm*MSTR address space to increase the effectiveness of the monitor.

Related tasks:

Determining z/OS Workload Manager velocity goals

System-provided service classes (MVS Planning: Workload Management)

Organizing work into workloads and service classes (MVS Planning: Workload Management)

Related reference:

MVS Planning: Workload Management (MVS Planning: Workload Management)

Related information:

DSNV508I (DB2 Messages)

DSNV512I (DB2 Messages)

Chapter 13. Limiting resources for a stored procedure

DB2 stored procedures are especially designed for high volume online transactions.

Procedure

To establish limits for stored procedures:

Take one of the following actions:

- Set a processor limit for each stored procedure, by updating the ASUTIME column of the SYSIBM.SYSROUTINES catalog table. This limit allows DB2 to cancel procedures that loop.
- Set a limit for the maximum number of times that a procedure can terminate abnormally, by specifying a value in the MAX ABEND COUNT field on installation panel DSNTIPX. This limit is a system limit that applies to all stored procedures and prevents a problem procedure from overwhelming the system with abend dump processing.
- Set a limit for the maximum number of times that a specific procedure can terminate abnormally, by specifying the STOP AFTER FAILURES option on the ALTER or CREATE PROCEDURE statement. This limit allows you to override the system limit specified in MAX ABEND COUNT and specify different limits for different procedures.

Related tasks:

Maximizing the number of procedures or functions that run in an address space

Related reference:

MAX ABEND COUNT field (STORMXAB subsystem parameter) (DB2 Installation and Migration)

CREATE PROCEDURE (DB2 SQL)

SYSIBM.SYSROUTINES table (DB2 SQL)

Chapter 14. Setting limits for system resource usage by using the resource limit facility

The DB2 *resource limit facility* (governor) enables you to limit resource usage, bind operations, and parallelism modes in the processing of certain SQL statements. Resource limits apply only to dynamic SQL statements.

About this task

Introductory concepts
The resource limit facility (Introduction to DB2 for z/OS)

You can use the resource limit facility for following activities:

- Set warning and error thresholds for SQL statements. The resource limit facility can inform users (through your application programs) that a processing limit might be exceeded for a particular SQL statement. These limits are sometimes called *predictive governing*. The resource limit facility provides an estimate of the processing cost of SQL statements before they run. To predict the cost of an SQL statement, you execute EXPLAIN to put information about the statement cost in DSN_STATEMNT_TABLE. If the statement exceeds a predictive governing limit, it receives a warning or error SQL code.
- These limits are sometimes called *reactive governing*.
- Restrict bind and rebind activities to avoid performance impacts on production data.

You can use reactive or predictive governing separately, or in combination.

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

Resource limits apply only to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Procedure

To set limits for system resource usage:

1. Create a DSNRLST*xx* table, a DSNRLMT*xx* table, or one of each.
2. Populate the content of the table or tables.
3. Use the START RLIMIT command to activate the resource limit facility.

Related tasks:
Specifying and changing resource limits

"Monitoring threads and connections by using profiles" on page 108

Related reference:

Resource limit facility tables

-START RLIMIT (DB2) (DB2 Commands)

-DISPLAY RLIMIT (DB2) (DB2 Commands)

Resource limit facility controls

System administrators can use the resource limit facility to limit the amount of time that is permitted for the execution of certain types SQL statements and bind operations.

Introductory concepts

The resource limit facility (Introduction to DB2 for z/OS)

GUPI Resource limits can apply only to the following types of SQL statements:

Subsystem parameters

The following subsystem parameters control the resource limit facility:

RLF
Specifies whether the resource limit facility starts automatically when DB2 starts.

RLFTBL
Specifies the identifier of the resource limit tables to be started automatically when DB2 starts, or when no ID value is specifed in a START RLIMIT command.

RLFERR
Specifies a default limit for dynamic SQL statements that originate from the local server.

RLFERRD
Specifies a default limit for dynamic SQL statements that originate from remote locations.

Commands

You can use the following commands for controlling the resource limit facility:

START RLIMIT
Starts the resource limit facility and identifies a resource limit specification table. You can also use the START RLIMIT command to switch resource limit specification tables.

STOP RLIMIT
Stops the resource limit facility and removes any set limits.

DISPLAY RLIMIT
Displays the current status of the resource limit facility. If the resource limit facility has been started, the output from the command also identifies the resource limit specification table.

Supplied user tables

The limits are defined in resource limit specification tables and can vary for different users. One resource limit table is used for each invocation of the resource limit facility and that table is specified in the START RLIMIT command.

The following user tables control the resource limit facility:

RLST*xx*
: Specify resource limits that apply based on the collection ID, package name, authorization ID, and location name of the SQL statement.

RLMT*xx*
: Specify resource limits that apply based on client information, including the application name, user ID, workstation ID, and IP address of the client

GUPI

Related tasks:

Managing resource limit tables

Related reference:

DSNTIPO: Operator functions panel (DB2 Installation and Migration)

Resource limit facility tables

Setting default resource limits for SQL statements

You can specify default resource limits that apply to certain categories of SQL statements when no resource limit table row applies to the statement.

About this task

You can use subsystem parameters to set default limits that apply to all SQL statements of certain types. The subsystem parameter values are used only if no resource limit table row applies to a statement.

Procedure

To set resource limits that apply to SQL statements by default:

1. Set the values of the appropriate subsystem parameter for the type of SQL statements that you want to limit:

Option	Description
Local dynamic SQL statements	Set the value of the RLFERR subsystem parameter.
Dynamic SQL statements from remote locations	Set the value of the RLFERRD subsystem parameter.

2. Issue a START RLIMIT command to start or restart the resource limit facility. You might need to first issue a STOP RLIMIT command if resource limits are already started for all resource limits that you want to apply.

Results

Each value specifies the action that DB2 takes for SQL statements when no row that applies to the statement is found in the resource limit table.

Related concepts:

Resource limit facility controls

Related reference:

-START RLIMIT (DB2) (DB2 Commands)

-STOP RLIMIT (DB2) (DB2 Commands)

DSNTIPO: Operator functions panel (DB2 Installation and Migration)

Specifying and changing resource limits

You can specify resource limits to be enforced by the resource limit facility by populating the resource limit tables with rows of data that describe the limits.

Before you begin

Create one or more resource limit tables.

About this task

Resource limits apply only to dynamic SQL statements. The resource limits are specified in supplied user tables that are named DSNRLST*xx* or DSNRLMT*xx*, where *xx* is a unique identifier.

| If both DSNRLMT*xx* and DSNRLST*xx* tables exist, rows in the DSNRLMT*xx* table that match a statement take priority over any matching rows in the DSNRLST*xx* table.

Procedure

Issue SQL statements, such as INSERT, UPDATE, MERGE, and DELETE statements, to populate the resource limit table. You can modify data in resource limit facility tables with only the usual table privileges. Higher authorities are not required.

Results

When the resource limit facility is started, changes to the resource limit specification table are immediately effective for all new threads. The changes also become effective for any existing threads that have not yet issued their first SQL statements of the following types:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

However, if you change the resource limit specification table while a thread is executing, the limit that existed when the thread issued its first SQL statement applies throughout the life of the thread, until DB2 reads in the new limit.

DB2 reads in a new limit in the following situations:

- When the application uses a different primary authorization ID.

- When the resource limit facility is stopped and started again.
- When a predictively governed package is loaded for execution.

Related tasks:

Managing resource limit tables

Related reference:

DSNRLMTxx resource limit tables

DSNRLSTxx resource limit tables

-START RLIMIT (DB2) (DB2 Commands)

-STOP RLIMIT (DB2) (DB2 Commands)

Limiting resources for SQL statements reactively

You can use the resource limit facility to set limits for *reactive governing*, which means that DB2 stops SQL statements from specified contexts that overuse system resources.

About this task

When you specify *reactive governing*, the resource limit facility stops a currently running SQL statement that meets the conditions in a resource limit table row when the SQL statement uses more than the maximum amount of resources specified by the value of the ASUTIME column in that row. When a statement exceeds a reactive governing limit, the application program receives SQLCODE -905. The application must include code that performs the appropriate action based on this situation.

Resource limits apply only to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements.

For statements that contain external user defined functions, the resource time used by the user defined functions is not counted as part of the resource time for the statement. No limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

Procedure

To specify reactive resource limits:

Specify either of the following values in the RLFFUNC column of the resource limit table:

'2' Limit dynamic SQL statements by package name, authorization ID, collection ID, the location name of the requester, or a combination of them. (RLST)

'8' Limit dynamic SQL statements by client information (RLMT)

Results

DB2 resets the accumulated ASUTIME at the following events:

- After the execution completes for an INSERT or UPDATE statement that includes a parameter marker.
- After the close of a cursor for a statement.
- During the PREPARE for dynamic statements.

Any statement that reaches or exceeds a limit that you set in a resource limit table terminates with SQLCODE -905 and the corresponding SQLSTATE '57014' . You can establish a single limit for all users, different limits for individual users, or both. Limits do not apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority. For queries that enter DB2 from a remote site, the local site limits are used.

For a dynamic statement that is monitored by the resource limit facility and that is inside a routine, the ASUTIME value that is specified for the top-level calling package is applied for the entire thread.

If the failed statement involves an SQL cursor, the cursor's position remains unchanged. The application can then close that cursor. All other operations with the cursor do not run and the same SQL error code occurs.

If the failed SQL statement does not involve a cursor, then all changes that the statement made are undone before the error code returns to the application. The application can either issue another SQL statement or commit all work done so far.

What to do next

Consider setting default resource limits that apply when resource limit tables cannot be accessed or matching resource limit table rows do not exist.For detailed information about creating default resource limits, see Setting default resource limits for SQL statements.

Related tasks:

Combining reactive and predictive governing

Limiting resources for SQL statements predictively

Calculating service unit values for resource limit tables

Related reference:

Resource limit facility tables

Related information:

-905 (DB2 Codes)

Limiting resources for SQL statements predictively

You can use the resource limit facility for *predictive governing* to avoid wasting processing resources by giving you the ability to prevent a SQL statement from running when it appears likely to exceed processing limits. In reactive governing, those resources are already used before the query is stopped.

About this task

The following figure provides an overview of how predictive governing works.

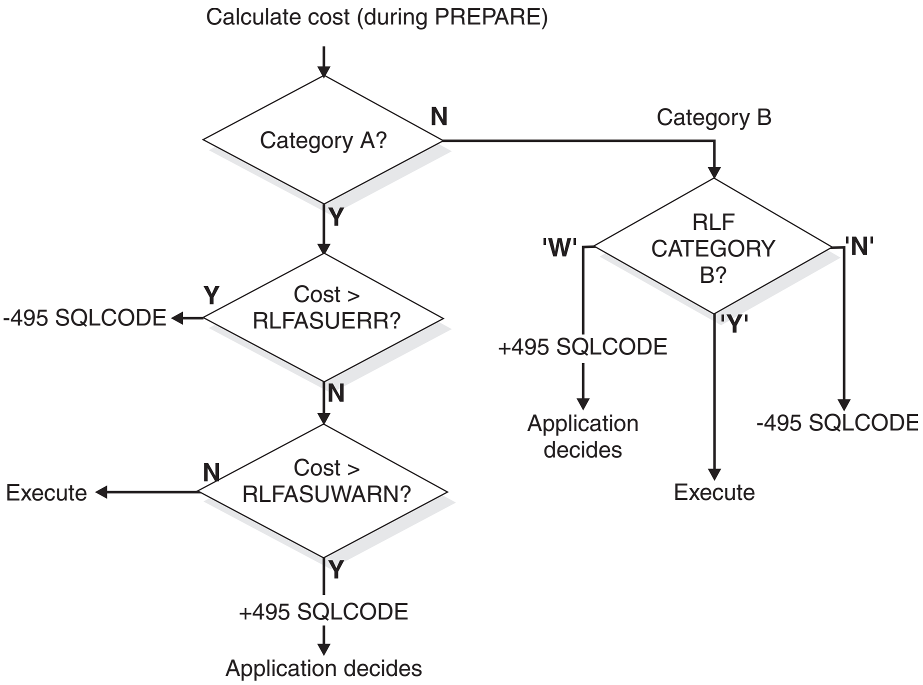


Figure 9. Processing for predictive governing

Resource limits apply only to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

At prepare time for a dynamic SQL statement, DB2 searches the active resource limit table to determine if the processor cost estimate exceeds the error or warning threshold that you set in the RLFASUWARN and RLFASUERR columns of the resource limit table. DB2 compares the cost estimate for a statement to the thresholds that you set, and the following actions occur:

- If the cost estimate is in cost category A and the error threshold is exceeded, DB2 returns a -495 SQLCODE to the application, and the statement is not prepared or run.
- If the estimate is in cost category A and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.
- If the estimate is in cost category B, DB2 takes the action that you specify in the RLF_CATEGORY_B column; that is, it either prepares and executes the statement, does not prepare or execute the statement, or returns a warning SQLCODE, which lets the application decide what to do.
- If the estimate is in cost category B and the value in the RLF_CATEGORY_B column is 'W', a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.

If SQLCODE +495 is returned to a down-level DRDA requester, OPEN processing continues but the first block of data is not returned with the OPEN. Thus, if your application does not continue with the query, you have already incurred the performance cost of OPEN processing.

For enabled requesters, if your application does not defer the prepare, SQLCODE +495 is returned to the requester and OPEN processing does not occur.

If your application does defer prepare processing, the application receives the +495 at its usual time (OPEN or PREPARE). If you have parameter markers with deferred prepare, you receive the +495 at OPEN time as you normally do. However, an additional message is exchanged.

Important: Do not use deferred prepare for applications that use parameter markers and that are predictively governed at the server side.

Procedure

To specify predictive governing:

Specify any of the following values in the RLFFUNC column of a resource limit table:

'7' Govern by package name (RLST)

'9' Govern by client information (RLMT)

Example

The following table is an RLST with two rows that use predictive governing.

Table 31. Predictive governing example

RLFFUNC	AUTHID	RLFCOLLN	RLFPKG	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
7	(blank)	COLL1	C1PKG1	900	1500	Y
7	(blank)	COLL2	C2PKG1	900	1500	W

The rows in the resource limit table for this example cause DB2 to act as follows for all dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements in the packages that are listed in this table (C1PKG1 and C2PKG1):

- Statements in cost category A that are predicted to be less than or equal to 900 SUs are executed.
- Statements in cost category A that are predicted to be greater than 900 and less than or equal to 1500 SUs receive a +495 SQLCODE.
- Statements in cost category A that are predicted to be greater than 1500 SUs receive SQLCODE -495, and the statement is not executed.

Related tasks:

Combining reactive and predictive governing

Limiting resources for SQL statements reactively

Related reference:

Resource limit facility tables

Related information:

+495 (DB2 Codes)

-495 (DB2 Codes)

Combining reactive and predictive governing

You can limit resource usage for certain SQL statements reactively during their execution, and you detect certain SQL statements that are likely to use too many resources predictively before they execute.

Example

Resource limits apply only to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

To use both reactive and predictive resource limits in combination, the resource limit table must contain at least two separate rows, as shown in the following table. If the processing cost estimate is in cost category B and you decide to run the statement, you can use a resource limit table to terminate the statement after a certain amount of processor time.

Table 32. Combining reactive and predictive governing

RLFFUNC	AUTHID	RLFPKG	ASUTIME	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
7	USER1	PKG2	0	800	1000	W
2	USER1	PKG2	1100	0	0	(blank)

The rows in the RLST resource limit table for this example cause DB2 to act as follows for a dynamic SQL statement that runs under the package named PKG2:

Predictive mode

- If the statement is in COST_CATEGORY A and the cost estimate is greater than 1000 SUs, USER1 receives SQLCODE -495 and the statement is not executed.
- If the statement is in COST_CATEGORY A and the cost estimate is greater than 800 SUs but less than 1000 SUs, USER1 receives SQLCODE +495.
- If the statement is in COST_CATEGORY B, USER1 receives SQLCODE +495.

Reactive mode

In either of the following cases, a statement is limited to 1100 SUs:

- The cost estimate for a statement in COST_CATEGORY A is less than 800 SUs

- The cost estimate for a COST_CATEGORY A is greater than 800 and less than 1000 or is in COST_CATEGORY B and the user chooses to execute the statement

Related tasks:

Limiting resources for SQL statements predictively

Limiting resources for SQL statements reactively

Related reference:

Resource limit facility tables

Related information:

+495 (DB2 Codes)

-495 (DB2 Codes)

Limiting resource usage for packages

You can specify limits for the amount of processor resources that are used by a specific group of SQL statements.

About this task

You can use a DSNRLST*xx* resource limit table with values that specify limits for resource usage by certain SQL statements, based on the following attributes:

- Collection
- Package name
- Authorization ID
- Location

Resource limits apply only to the following SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

Procedure

1. Insert rows into a DSNRLST*xx* resource limit table with values that identify the context of the governed statements, the type of governing, thresholds for predictive governing and limits for reactive governing.
   - To insert a row for reactive governing for dynamic SQL statements, specify '2' in RLFFUNC column and the amount of processor limit in ASUTIME column. PLANNAME must contain blank. The following table shows an example of reactive governing for dynamic SQL statements:

Table 33. Qualifying rows for reactive governing for dynamic SQL statements

RLFFUNC	AUTHID	PLANNAME	RLFCOLLN	RLFPKG	LUNAME	ASUTIME
2	JOE	(blank)	COL1	(blank)	(blank)	(null)
2	JOE	(blank)	COL2	PKG1	(blank)	15000
2	(blank)	(blank)	(blank)	PKG2	PUBLIC	10000

The first row in the table above shows that when a user, JOE, runs any package in a collection, COL1, at the local location, no limits restrict any dynamic statement in the package. The second row shows that when a user, JOE, runs a package, PKG1, in collection, COL2, at the local location, each dynamic statement in the package is restricted to 15,000 SUs. The third row shows that when any user runs a package, PKG2, in any collection from any location in the network, including the local location, a processor limit of 10,000 SUs is applied for each dynamic statement in the package.

- To insert a row for predictive governing of dynamic SQL statements, specify '7' in RLFFUNC column and the limit threshold values in RLFASUERR and RLFASUWARN columns. PLANNAME must contain blank. The following table shows an example of predictive governing:

Table 34. Qualifying rows for predictive governing

						RLF	RLF	RLF_
		PLAN	RLF			ASU	ASU	CATEGORY_
RLFFUNC	AUTHID	NAME	COLLN	RLFPKG	LUNAME	WARN	ERR	B
7	JOE	(blank)	COL1	PKG1	(blank)	7000	12000	W
7	(blank)	(blank)	(blank)	PKG2	PUBLIC	5000	9000	Y
7	PETER	(blank)	(blank)	(blank)	0	0	0	N

The first row in the table shows that when the user whose authorization ID is JOE runs a package at the local location, that a warning threshold is specified at 7000 SUs and an error threshold is specified at 12,000 SUs. The warning and error are applied when DB2 estimates that the amount of processor time consumed by a dynamic statement exceeds the specified thresholds. That row also specifies that a warning issues when for cost category B estimates.

The second row shows that when any user runs dynamic statements from the PGK2 package in any collection from any location in the network, including at the local location, that a warning threshold is specified at 5000 SUs and that an error threshold is specified at 9000 SUs. The statement is allowed to run if the estimate is based on cost category B.

The third row shows that when the user whose authorization ID is PETER runs any package in any collection at the local location, no dynamic statement is allowed to run even when the estimate is based on cost category B.

2. Issue the following command, where *xx* is the two character identifier that you specified when you created the table:

GUPI

```
-START RLIMIT ID=xx
```

You can start and stop different resource limit tables at different times. However, only one resource limit table of each type (DSNRLMT*xx* or DSNRLST*xx*) can run at any one time.

GUPI

Results

DB2 uses the following search order:
1. Exact match
2. Authorization ID
3. Collection ID and package name
4. LU name
5. No row match

When multiple rows that contain the same values in all other columns, the best matching row is chosen based on LU name in the following order for reactive governing, if any row exists:

Local agents

1. Matching LU name value
2. Blank value
3. 'PUBLIC'

Distributed agents

1. Matching LU name value
2. 'PUBLIC'

For predictive governing, the qualified row that appears first in the index order is chosen.

When no row in the resource limit table matches the currently executing statement, DB2 uses the default values that are specified by certain subsystem parameters. For information about the subsystem parameters that apply, see "Setting default resource limits for SQL statements" on page 157. The default limits apply to reactive governing only. For predictive governing, when no row matches, no predictive governing occurs.

When an SQL statement contains an external user-defined function, the execution time for the user-defined function is not included in the ASUTIME of the statement execution. The ASUTIME for an external execution of a user-defined functions is controlled based on the ASUTIME specified for the user-defined function in the CREATE FUNCTION statement.

What to do next

Consider setting default resource limits that apply when matching resource limit table rows do not exist.

Related concepts:

Cost categories

Related reference:

DSNRLSTxx resource limit tables

-START RLIMIT (DB2) (DB2 Commands)

DSNTIPR: Distributed data facility panel 1 (DB2 Installation and Migration)

Limiting resource usage by client information

You can limit the amount of processor resources that are used by a specific group of SQL statements based on the client information of the statements.

Before you begin

Provide client information to DB2 by using the appropriate application programming interface to set the values of the following special registers:

- CURRENT CLIENT_APPLNAME (DB2 SQL)
- CURRENT CLIENT_USERID (DB2 SQL)
- CURRENT CLIENT_WRKSTNNAME (DB2 SQL)

This action is required only when the client application does not use the default values.

About this task

You can use a DSNRLMT*xx* resource limit tables to specify limits and thresholds for resource usage that apply to certain SQL statements that run on middleware servers based on the following types of client information:

- Application name
- End-user ID
- Workstation ID
- IP address

The values for the client information that are found at the start of a new unit-of-work are used to determine which row of the RLMT table controls the processing of dynamic requests for the duration of the unit-of-work.

Resource limits apply only to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

If both DSNRLMT*xx* and DSNRLST*xx* tables exist, rows in the DSNRLMT*xx* table that match a statement take priority over any matching rows in the DSNRLST*xx* table.

Procedure

To limit the use of resources for middleware servers:

1. Insert values that identify the statements to limit, the type of governing, and the thresholds or limits, into a DSNRLMT*xx* resource limit table.

   The following table shows example rows that specify client-based limits for dynamic SQL statements.

Table 35. Qualifying rows for dynamic SQL statements for middleware servers

RLFFUNC	RLFEUAN	RLFEUID	RLFEUWN	RLFIP	ASUTIME
8	APP1	PAUL	(blank)	9.30.72.223	12000
8	(blank)	(blank)	WORKSTA10	(blank)	7000

   The first row in the table shows that when PAUL runs the APP1 application from the 9.30.72.223 IP address, the resource limit facility limits dynamic SQL statements run by APP1 to 12,000 SUs each. The last row shows that any dynamic SQL statements that are issued from work station 10 are limited to 7000 SUs each.

2. Issue this command:

   GUPI

   ```
   -START RLIMIT ID=xx
   ```

   xx is the two-character identifier that you specified when you created the table. You can start and stop different resource limit tables at different times. However, only one resource limit table of each type (DSNRLMT*xx* or DSNRLST*xx*) can run at any one time.

   GUPI

Results

DB2 uses the following search order:

1. Exact match
2. Application name
3. User ID
4. Workstation name
5. IP address
6. No row match

What to do next

Consider setting default resource limits that apply when resource limit tables cannot be accessed or matching resource limit table rows do not exist.For detailed information about creating default resource limits, see Setting default resource limits for SQL statements.

Related tasks:

Providing extended client information to the data source with IBM Data Server Driver for JDBC and SQLJ-only methods (DB2 Application Programming for Java)

Related reference:

DSNRLMTxx resource limit tables

-START RLIMIT (DB2) (DB2 Commands)

WLM_SET_CLIENT_INFO stored procedure (DB2 SQL)

DSNTIPO: Operator functions panel (DB2 Installation and Migration)

DSNTIPR: Distributed data facility panel 1 (DB2 Installation and Migration)

sqle_client_info data structure

sqleseti API - Set client information

Related information:

RLF enhancements (DB2 9 for z/OS Technical Overview)

Limiting resources for statements from remote locations

Several important guidelines and restrictions apply when you use the resource limit facility in a distributed processing environment.

About this task

Introductory concepts

Distributed data access (Introduction to DB2 for z/OS)
Remote DB2 access (Introduction to DB2 for z/OS)
Effects of distributed data on planning (Introduction to DB2 for z/OS)

Resource limits apply only to the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements.

Procedure

To use the resource limit facility in conjunction with distributed processing:

Remember the following guidelines:

Option	Description
Dynamic statements from requesters that use DRDA protocols using TCP/IP	You can create a DSNRLMT*xx* table to limit resources by client information (RLFFUNC='8' or RLFFUNC='9') such as: • Application name • User ID • Workstation ID • IP address You can use a DSNRLST*xx* table for this purpose. However, you must then govern by package name (RLFFUNC='2' or RLFFUNC='7')(RLFFUNC='2' or RLFFUNC='7'). In this case, you must also: • Specify a blank value for PLANNAME column. • Specify 'PUBLIC' for the value of the LUNAME column for all remote LUs. You cannot specify the LUNAME of the requester. The 'PUBLIC' value also applies to local locations. If the value of the LUNAME column is blank, DB2 assumes that the row applies only to the local location, and the row does not apply to any incoming distributed requests.
SQL statements from requesters that use DRDA protocols using SNA	You can use a DSNRLMT*xx* table to govern by the following types of client information (RLFFUNC='8' or RLFFUNC='9'): • Application name • User ID • Workstation ID However, IP address cannot be used, because no IP address is provided to DB2 under this DRDA protocol. You can use a DSNRLST*xx* table for this purpose. However, you must then govern by package name (RLFFUNC='2' or RLFFUNC='7'). In this case, you must also: • Specify a blank value for PLANNAME column. • Specify the LU name of the requestor or 'PUBLIC' for the value of the LUNAME column. A 'PUBLIC' value in the LUNAME column also applies to local locations. If the value of the LUNAME column is blank, DB2 assumes that the row applies only to the local location, and the row does not apply to any incoming distributed requests.

If no qualified row is present in the resource limit table to limit access from remote locations, the limit is controlled by the value of the RLFERRD subsystem parameter.

Related tasks:

Limiting resource usage by client information

Managing resource limit tables

Related reference:

DSNRLMTxx resource limit tables

DSNRLSTxx resource limit tables

RLST ACCESS ERROR field (RLFERRD subsystem parameter) (DB2 Installation and Migration)

DSNTIPR: Distributed data facility panel 1 (DB2 Installation and Migration)

Calculating service unit values for resource limit tables

The processing time for a particular SQL statement varies according to the processor that executes it, but the number of service units required is a relative metric, which remains roughly constant.

About this task

The resource limit facility samples processing time in a relative measurement called *service units*.

A relative metric is used so that you don't have to modify the resource limit table values when processors are changed. The number of service units consumed is not exact between different processors because the calculations for service units are dependent on performance averages measured before the release of a new processor. In some cases, DB2 workloads can differ from the measured averages. In these cases, resource limit table value changes might be necessary.

Procedure

To choose service unit times for the ASUTIME, RLFASUWARN, or RLFASUERR columns, use one of the following approaches:

- Use the value in the PROCSU column of DSN_STATEMNT_TABLE as your starting point. You can also get the value from the IFCID 0022 record.
- If you do not have statement table information, or if you have queries for which you have no information, you can use the following formula to calculate SU time:

  ```
  SU time = processor time
  × service units per second value
  ```

  The value for service units per second depends on the processor model. To find this value for your processor model, see Preparing an initial OPT (MVS Initialization and Tuning Guide).

Example

For example, if processor A is rated at 900 service units per second and you do not want any single SQL statement to use more than 10 seconds of processor time, you could set ASUTIME by using the following calculation:

```
ASUTIME time = 10 seconds
× 900 service units/second = 9000 service units
```

Later, you might upgrade to processor B, which is rated at 1000 service units per second. If the value you set for ASUTIME remains the same (9000 service units), the SQL statement is only allowed 9 seconds for processing time but an equivalent number of processor service units:

```
ASUTIME = 9 seconds
× 1000 service units/second = 9000 service units
```

As this example illustrates, after you establish an ASUTIME (or RLFASUWARN or RLFASUERR) for your current processor, you do not need to modify it when you change processors.

Related tasks:

Managing resource limit tables

Related reference:

Resource limit facility tables

DSN_STATEMNT_TABLE

Restricting bind operations

You can use the DB2 resource limit facility to set limits on the amount of resources used by bind operations.

Procedure

To restrict bind operations:

Use an RLST resource limit table and specify RLFFUNC='1' and qualify rows by authorization ID and LU name. The same precedence search order applies:

1. AUTHID and LUNAME match.
2. AUTHID matches.
3. LUNAME matches.

   A value of PUBLIC for LUNAME applies to all authorization IDs at all locations, while a blank LUNAME governs bind operations for IDs at the local location only.
4. If no entry matches, or if your resource limit table cannot be read, the resource limit facility does not disable bind operations.

Example

The table below is an example of an RLST resource limit table that disables bind operations for all but three authorization IDs. Notice that the binder from the local site is able to bind but that the binder from San Francisco is not able to bind. Everyone else from all locations, including the local one, is disabled from processing binds.

Table 36. Restricting bind operations

RLFFUNC	AUTHID	LUNAME	RLFBIND
1	BINDGUY	PUBLIC	
1	NIGHTBND	PUBLIC	
1	(blank)	PUBLIC	N
1	BINDER	SANFRAN	N
1	BINDER	(blank)	

Related reference:

DSNRLSTxx resource limit tables

Managing resource limit tables

You can use *resource limit tables* to specify resource usage limits for SQL statements.

About this task

Introductory concepts

The resource limit facility (Introduction to DB2 for z/OS)

Resource limit tables enable you to limit the amount of processor resources, in service units, that are used by the following types of SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

The DSNTIJSG installation job contains statements that create the database, table space, tables, and indexes for the resource limit facility. You can tailor those statements to create or update the format of resource limit tables.

To create a new resource limit specification table, you must have sufficient authority to define objects in the DSNRLST database and to specify *authid*, which is the authorization ID specified in the value of the RLFAUTH subsystem parameter.

You can create two different types of resource limit tables, *resource limit specification tables* (named DSNRLST*xx*) and *resource limit middleware tables* (named DSNRLMT*xx*). You can create instances of either type of table, or instances of both, depending on your specific plan for limiting resources.

If you are a system administrator, you must determine how your location intends to use the resource limit facility and create several local procedures:

- For creating and maintaining your resource limit tables
- For establishing limits for any newly written applications
- For console operators, such as switching resource limit tables every day at a certain time

Resource limit tables can reside in any database. However, because a database has some special attributes while the resource limit facility is active, it is best to create resource limit tables in their own database. Only one resource limit table can be active at any particular time, however you might create several instances of either or both types of resource limit tables and use different instances at different times.

Related concepts:

Resource limit facility controls

Related tasks:

Setting limits for system resource usage by using the resource limit facility

Installation step 17: Define and bind DB2 objects: DSNTIJSG (DB2 Installation and Migration)

Related reference:

Resource limit facility tables

RESOURCE AUTHID field (RLFAUTH subsystem parameter) (DB2 Installation and Migration)

Creating resource limit tables

You can create resource limit tables and related indexes to limit the amount of resources that are used for processing SQL statements.

About this task

You can create different types of resource limit tables, depending on context of the queries that you want to limit.

If both DSNRLMT*xx* and DSNRLST*xx* tables exist, rows in the DSNRLMT*xx* table that match a statement take priority over any matching rows in the DSNRLST*xx* table.

Procedure

Create the type of resource limit table and related index in the DSNRLST database. Choose the type of table according to the context of the queries that you want to govern, and how you will identify those queries: The DSNTIJSG installation job contains statements that create the database, table space, tables, and indexes for the resource limit facility. You can tailor those statements to create or update the format of resource limit tables.

Option	Description
Plan name, collection ID, package name, authorization ID, and location name:	1. Issue the create table statement for *authid*.DSNRLST*xx*, where *xx* is two alphanumeric characters that identifies the particular instance of the table. **Important:** The value of *xx* must be alphanumeric and cannot contain special or double-byte characters. 2. Issue the create index statement for *authid*.DSNARL*xx*, where *xx* is the same two alphanumeric characters that identify the table.

Option	Description
Client information, including application name, user ID, workstation ID, and IP address:	1. Issue the create table statement for *authid*.DSNRLMT*xx*, where *xx* is two alphanumeric characters that identifies the particular instance of the table. **Important:** The value of *xx* must be alphanumeric and cannot contain special or double-byte characters. 2. Issue the create index statement for *authid*.DSNMRL*xx*, where *xx* is the same two alphanumeric characters that identify the table.

Related reference:

DSNRLMTxx resource limit tables

DSNRLSTxx resource limit tables

Starting and stopping resource limit tables

You can create several resource limit tables and start and stop them as needed to support the type and amount of processing that occurs at different times.

About this task

At installation time, you can specify a default set of resource limit tables to be used each time that DB2 is restarted. The RLF subsystem parameter value controls the automatic start, and the value of the RLFTBL subsystem parameter specifies the tables to start when DB2 starts.

If the resource limit facility is active and you restart it without stopping it, any jobs that are active continue to use their original limits, and all new jobs use the limits that are specified in the new table.

If you stop the resource limit facility while a statement is running, it runs with no limit, but its processing time continues to accumulate. If you later restart the resource limit facility, the new limit takes effect for an active job only when the job passes one of several internal checkpoints. For example, an SQL statement that builds a result table and fetches from it, passes those checkpoints at intervals that might range from moments to hours. As a result, a change to a resource limit might not stop an active statement within the time that you expect.

Only one pair of resource limit tables, containing one table of each type (DSNRLST*xx* or DNSRLMT*xx*), can be active at any time.

Procedure

To start and stop limit tables:

- Start resource limit tables by issuing START RLMIT commands. For example, you can issue the following command:

  GUPI

  ```
  -START RLIMIT ID=xx
  ```

  xx is the two-character identifier in the names of the set of tables that you want to start (DSNRLST*xx*, DNSRLMT*xx*, or both if they exist).

GUPI

The specified limits apply to all subsequent threads.

- When the resource limit facility is already active, restart the resource limit facility issuing START RLIMIT commands and specifying the identifier of an inactive set of tables . The resource limit facility is re-started and the limits in the specified tables are applied to all subsequent threads.
- Issue DISPLAY RLIMIT commands to determine the active set of resource limit tables.
- Stop resource limits by issuing STOP RLIMIT commands. For example, you can issue the following command:

  GUPI

  ```
  -STOP RLIMIT ID=xx
  ```

  xx is the two-character identifier in the names of the set of tables that you want to stop.

  GUPI

  All resource limits are stopped.
- Issue CANCEL THREAD commands to stop an active job that does not pick up the new limit when you restart the resource limit facility.

Example

You can use different resource limit tables for the day shift and the night shift. Assume that the DSNRLST01 and DSNRLST02 resource limit tables contain the following rows.

Table 37. Example resource limit table values for the day shift, in DSNRLST01

AUTHID	PLANNAME	LUNAME
BADUSER		LUDBD1
ROBYN		LUDBD1
		LUDBD1

Table 38. Example resource limit table values for the night shift, in DSNRLST02

AUTHID	ASUTIME	LUNAME
BADUSER	0	LUDBD1
ROBYN	NULL	LUDBD1
	50000	LUDBD1

In this example, you might issue the following command when the day shift begins:

GUPI

```
-START RLIMIT ID=01
```

GUPI

Then, when it is time for the night shift to begin, you might issue the following command to switch to the other table:

GUPI

```
-START RLIMIT ID=02
```

GUPI

After you issue this command, new threads from the ROBYN authorization ID from LUDBD1 can issue SQL statements without limits. Threads that already existed at the time that you issued the command continue under the limits that are specified in the DSNRLST01 table.

Related concepts:

Boolean term predicates

How DB2 simplifies join operations

Related reference:

-START RLIMIT (DB2) (DB2 Commands)

-DISPLAY RLIMIT (DB2) (DB2 Commands)

-STOP RLIMIT (DB2) (DB2 Commands)

Restricted activity on resource limit tables

While the resource limit facility is active, you cannot execute certain SQL statements on the resource limit tables, or the table space and database that contain the resource limit table

The statements that are restricted while the resource limit facility is active are:

- DROP DATABASE
- DROP INDEX
- DROP TABLE
- DROP TABLESPACE
- RENAME TABLE

You cannot stop a database or table space that contains an active resource limit table; nor can you start the database or table space with ACCESS(UT).

Related tasks:

Specifying and changing resource limits

Dropping tables (DB2 Application programming and SQL)

Dropping DB2 databases (DB2 Administration Guide)

Dropping and redefining a DB2 index (DB2 Administration Guide)

Dropping, re-creating, or converting a table space (DB2 Administration Guide)

Related reference:

Resource limit facility tables

DROP (DB2 SQL)

RENAME (DB2 SQL)

Chapter 15. Reducing processor resource consumption

Many factors affect the amount of processor resources that DB2 consumes.

Procedure

To reduce the consumption of processor resources by DB2:

- Cache authorizations for plans, packages, and routines, such as user-defined functions and stored procedures.
- Use an isolation level of cursor stability and CURRENTDATA(NO) to allow lock avoidance.
- Use the LOCKSIZE clause, when you create or alter a table space. Doing so avoids using excess granularity for locking, such as row-level locking.
- Reorganize indexes and table spaces to improve the performance of access paths. Reorganizing indexes and table spaces is also important after table definition changes, such as changing the data type of a column, so that the data is converted to its new definition. Otherwise, DB2 must track the data and apply the changes as the data is accessed.
- Ensure that your access paths are effective by:
  - Creating only necessary indexes
  - Updating statistics regularly
  - Rebinding as necessary

Related tasks:

Choosing an ISOLATION option

Maintaining DB2 database statistics

Deciding whether to rebind after you collect statistics

Maintaining data organization

Specifying the size of locks for a table space

Reusing threads for your high-volume transactions

For high-volume transactions, reusing threads can help performance significantly.

About this task

By reusing threads, you can reduce the amount processor resources that DB2 consumes.

Procedure

To reuse threads:

- For IMS, process multiple input messages in one scheduling of the IMS processing program
  - Set PROCLIM to a value greater than 1 and use class priority scheduling to share the cost of thread creation and termination among more than one transaction.
  - Reuse threads with wait for input (WFI), or the IMS fast path and class scheduling.

- For CICS, enhance thread reuse through specifications for pool and entry threads in the CICS resource definition online (RDO). You can useResource Recovery Services attachment facility to reuse threads.

Related concepts:

Resource Recovery Services attachment facility (DB2 Application programming and SQL)

Related tasks:

Invoking the Resource Recovery Services attachment facility (DB2 Application programming and SQL)

Minimizing the processing cost of DB2 traces

By suppressing DB2 trace options, you might significantly reduce processing costs.

About this task

The DB2 trace facility can consume a large amount of processing resources. Performance trace and global trace are especially resource-intensive.

Procedure

To reduce the cost of processing for the DB2 trace facility:

- Enable only the minimal trace and audit classes that you need. You can enable more detailed traces only when you encounter specific performance problems.
- Turn off global trace to significantly reduce processor consumption. Global trace requires 2% to 100% more processor usage. If possible, turn off DB2 global trace. To turn off global trace:
  1. Specify NO for the TRACSTR subsystem parameter.
  2. If the global trace becomes needed for serviceability, use the START TRACE command to start it.
- Avoid activating accounting class 2 if possible.

  Enabling accounting class 2 along with accounting classes 1 and 3 provides more detail that relates directly to the accounting record IFCID 0003, and records thread level entry into and exit from DB2. By activating class 2, you can separate DB2 times from application times.

  Running accounting class 2 adds to the cost of processing. How much extra cost depends on how much SQL the application issues. Typically, an online transaction incurs an extra 2.5% when it runs with accounting class 2. A typical batch query application, which accesses DB2 more often, incurs about 10% extra cost when it runs with accounting class 2. If most of your work is through CICS, you most likely do not need to run with class 2 because the class 1 and class 2 times are very close.

  However, if you use CICS Transaction Server for z/OS with the Open Transaction Environment (OTE), activate and run class 2. If you are concerned about high volume DB2 accounting records, for the DDF or DRDA threads and RRS attach threads, you can reduce the number of DB2 accounting records by using the ACCUMACC subsystem parameter, which consolidates multiple accounting records into one.
- Consider the cost of audit trace. When the audit trace is active, the more tables that are audited and the more transactions that access them, the greater the performance impact. The cost of audit trace is typically less than 5%.

When you estimate the performance impact of the audit trace, consider the frequency of certain events. For example, security violations are not as frequent as table accesses. The frequency of utility runs is likely to be measured in executions per day. Alternatively, authorization changes can be numerous in a transaction environment.

- Turn on performance trace classes only for specific performance problems. The combined cost of all performance classes runs from about 20% to 100%. The extra cost for performance trace classes 1 - 3 is typically in the range of 5% to 30%. It is best to turn on only the performance trace classes that address a specific performance problem and qualify the trace as much as possible to limit the data that is gathered to only the data that you need.
- Specify appropriate constraints and filters when you start traces. By doing so, you can limit the collection of trace data to particular applications or users and to limit the data that is collected to particular traces and trace events. You can use *trace constraints* to limit the scope of the collected data to a particular context and to particular traces and trace events. Similarly, you can use *trace filters* to exclude the collection of trace data from specific contexts and to exclude the collection of specific traces and trace events.

  For example, you can specify constraints and filters by application and user attributes such as collection ID, package name, location name, workstation name, authorization ID, user ID, role, and more. You can also use constraints and filters to limit the collection of trace data to certain trace classes and particular trace events (IFCIDs). For a complete list of the available constraint and filter options, see -START TRACE (DB2) (DB2 Commands).
- Use the STATIME subsystem parameter to control the interval for writing IFCID 105 and 106 records from statistics class 1. Starting statistics traces class 1,3,4 (and 5 for data sharing environments) provides data at the system level. Because statistics trace information is written only periodically, CPU cost is negligible.

What to do next

Suppressing other trace options, such as TSO, IRLM, z/OS, IMS, CICS, and other trace options, can also reduce costs.

Related concepts:

DB2 trace

Types of DB2 traces

Related tasks:

Minimizing the volume of DB2 trace data

Related reference:

TRACE AUTO START field (TRACSTR subsystem parameter) (DB2 Installation and Migration)

STATISTICS TIME field (STATIME subsystem parameter) (DB2 Installation and Migration)

DDF/RRSAF ACCUM field (ACCUMACC subsystem parameter) (DB2 Installation and Migration)

-START TRACE (DB2) (DB2 Commands)

-MODIFY TRACE (DB2) (DB2 Commands)

-STOP TRACE (DB2) (DB2 Commands)

-DISPLAY TRACE (DB2) (DB2 Commands)

Part 4. Improving concurrency

You can make better use of your resources and improve concurrency by understanding the effects of the parameters that DB2 uses to control locks

Before you begin

Some performance problems might seem to be locking problems even though they are really problems somewhere else in the system. For example, a table space scan of a large table can result in timeout situations. Similarly, when tasks are waiting or swapped out, and the unit of work is not committed, the tasks continue to hold locks. When a system is heavily loaded, contention for processing, I/O, and storage can also cause waiting.

Therefore, You might consider the following approaches before you take specific actions to tune locks:

- Resolve overall system, subsystem, and application performance problems to ensure that you not only eliminate locking symptoms, but also correct other underlying performance problems.
- Reduce the number of threads or initiators.
- Increase the priority of DB2 tasks on the system.
- Increase the amount of processor resources, I/O, and real memory.

About this task

You might not need to do anything about DB2 locks. Explicit lock requests are not necessary to prevent concurrent applications from reading or modifying uncommitted data. Applications acquire implicit locks under the control of DB2 to preserve data integrity. However, locks can sometimes result in performance problems from contention situations, such as *suspension*, *timeout*, and *deadlock*.

You can sometimes prevent such situations by considering concurrency when you design your system and subsystem options, databases, and applications.

Procedure

To achieve acceptable concurrency in your DB2 subsystems, you can follow certain basic recommendations. The recommendations described here are basic starting points for improving concurrency. Therefore, detailed analysis of your data design and applications might be required to achieve the best possible concurrency:

- Bind most applications with the ISOLATION(CS) and CURRENTDATA(NO) options. These options enable DB2 to release locks early and avoid taking locks in many cases.
- Use the REORG utility to keep your data organized. Doing so can prevent the additional lock and unlock requests for situations such as updates to compressed and varying-length rows, and auto-release locks for pseudo-deleted index entries and keys.
- Use LOCKSIZE ANY or PAGE as a design default. Consider LOCKSIZE ROW only when applications encounter significant lock contention, including deadlock and timeout.

LOCKSIZE ANY is the default for CREATE TABLESPACE. It allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE and LOCKMAX SYSTEM for non-LOB/non-XML table spaces. For LOB table spaces, DB2 chooses LOCKSIZE LOB and LOCKMAX SYSTEM. Similarly, for XML table spaces, DB2 chooses LOCKSIZE XML and LOCKMAX SYSTEM.

Page-level locking generally results in fewer requests to lock and unlock data for sequential access and manipulation, which translates to reduced CPU cost. Page-level locking is also more likely to result in sequentially inserted rows in the same data page. Row-level locking with MAXROWS=1 can suffer from data page p-locks in data sharing environments. However, page-level locking can avoid the data page p-locks when MAXROWS=1.

Row-level locking provides better concurrency because the locks are more granular. However, the cost of each lock and unlock request is roughly the same for both page and row-level locking. Therefore, row-level locking is likely to incur additional CPU cost. Row-level locking might also result in more data page latch contention. Sequentially inserted rows, by concurrent threads, are less likely to be in the same data page under row-level locking.

- Reduce locking contention on the catalog and directory for data definition, bind, and utility operations You can use the following approaches to reduce this type of contention:
  - Reduce the number of objects per database.
  - Group data definition statements from the same database within the same commit scope, apart from data manipulation statements, and commit frequently.
  - Assign a unique authorization ID and private database to each user.
  - Avoid using LOCK TABLE statements and statements that use RR isolation to query the catalog.
- Specify the TRACKMOD NO and MEMBER CLUSTER options when you create table spaces. These options can reduce p-lock and page latch contention on space map pages during heavy inserts into GBP-dependent table spaces. TRACKMOD NO cannot be used when incremental image copies are used for the table spaces.
- Use the RELEASE(DEALLOCATE) bind option to avoid the cost of repeatedly releasing and reacquiring locks for applications that use frequent commit points for repeated access to the same table spaces.
- Use the RELEASE(COMMIT) bind option for plans or packages that are used less frequently to avoid excessive increases to the EDM pool storage.
- For mixed INSERT, UPDATE, and DELETE workloads consider the LOCKSIZE PAGE and MAXROWS 1 options to reduce page latch contention on data pages. Do not use LOCKSIZE ROW for such mixed workloads, regardless of whether MEMBER CLUSTER is used. MAXROWS 1 is recommended only when high levels of lock or latch contention are encountered. The trade-off is a potential increase in getpage and read-writer I/O operations. The number of pages required to contain the data might increase by as many rows as can fit on a page when MAXROWS 1 is used. For example, if 20 rows fit in a single page, then the result is a 20 times increase in the number of pages used. Another result is a significantly reduce buffer pool hit ratio.

What to do next

For DB2 subsystems that are members of data sharing groups extra recommendations apply. For information about improving concurrency in data sharing groups, see Improving concurrency in data sharing environments (DB2 Data Sharing Planning and Administration).

PSPI

Related concepts:

Locks and Latches (DB2 for z/OS Best Practices)

Related tasks:

Configuring subsystems for concurrency

Designing databases for concurrency

Programming for concurrency

Monitoring concurrency and locks

Analyzing concurrency

Improving concurrency for real-time statistics data

Chapter 16. Concurrency and locks

Concurrency is the ability of more than one application process to access the same data at essentially the same time.

PSPI

An application for order entry is used by many transactions simultaneously. Each transaction makes inserts in tables of invoices and invoice items, reads a table of data about customers, and reads and updates data about items on hand. Two operations on the same data, by two simultaneous transactions, might be separated only by microseconds. To the users, the operations appear concurrent.

Why DB2 controls concurrency

Concurrency must be controlled to prevent lost updates and such possibly undesirable effects as unrepeatable reads and access to uncommitted data.

Lost updates
Without concurrency control, two processes, A and B, might both read the same row from the database, and both calculate new values for one of its columns, based on what they read. If A updates the row with its new value, and then B updates the same row, A's update is lost.

Access to uncommitted data
Also without concurrency control, process A might update a value in the database, and process B might read that value before it was committed. Then, if A's value is not later committed, but backed out, B's calculations are based on uncommitted (and presumably incorrect) data.

Unrepeatable reads
Some processes require the following sequence of events: A reads a row from the database and then goes on to process other SQL requests. Later, A reads the first row again and must find the same values it read the first time. Without control, process B could have changed the row between the two read operations.

To prevent those situations from occurring unless they are specifically allowed, DB2 might use *locks* to control concurrency.

How DB2 uses locks

A lock associates a DB2 resource with an application process in a way that affects how other processes can access the same resource. The process associated with the resource is said to “hold” or “own” the lock. DB2 uses locks to ensure that no process accesses data that has been changed, but not yet committed, by another process. For XML and LOB locks, DB2 also uses locks to ensure that an application cannot access partial or incomplete data

Locks might cause *contention*, which degrades performance, including situations such as suspensions, timeouts, and deadlocks.

Related concepts:

Lock contention

Transaction locks

Locks for LOB data

Locks for XML data

Related tasks:

Configuring subsystems for concurrency

Designing databases for concurrency

Programming for concurrency

Lock contention

Locks are important for maintaining concurrency in the DB2 environment. However, locks might cause several types of *contention* situations that degrade DB2 performance, including suspension, timeout, and deadlock.

PSPI

Suspension

An application encounters *suspension* when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running. A suspended process resumes when all processes that hold the conflicting lock release them or the requesting process experiences a timeout or deadlock and the process resumes and handles an error condition.

Incoming lock requests are queued. Requests for lock promotion, and requests for a lock by an application process that already holds a lock on the same object, precede requests for locks by new applications. Within those groups, the request order is “first in, first out.”

For example, using an application for inventory control, two users attempt to reduce the quantity on hand of the same item at the same time. The two lock requests are queued. The second request in the queue is suspended and waits until the first request releases its lock.

The suspended process resumes running when:

- All processes that hold the conflicting lock release it.
- The requesting process times out or deadlocks and the process resumes to deal with an error condition.

Timeout

An application process encounters a *timeout* when it terminates because of a suspension that exceeds a preset interval. DB2 terminates the process, issues messages, and returns error codes.

For example, an application process attempts to update a large table space that is being reorganized by the utility REORG TABLESPACE with SHRLEVEL NONE. It is likely that the utility job does not release control of the table space before the application process times out.

DB2 terminates the process, issues two messages to the console, and returns SQLCODE -911 or -913 to the process (SQLSTATEs '40001' or '57033'). Reason code

00C9008E is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0196.

If you are using IMS, and a timeout occurs, the following actions take place:

- In a DL/I batch application, the application process abnormally terminates with a completion code of 04E and a reason code of 00D44033 or 00D44050.
- In any IMS environment except DL/I batch:
  - DB2 performs a rollback operation on behalf of your application process to undo all DB2 updates that occurred during the current unit of work.
  - For a non-message driven BMP, IMS issues a rollback operation on behalf of your application. If this operation is successful, IMS returns control to your application, and the application receives SQLCODE -911.

    If the operation is not successful, the response depends on the value of GSROLBOK parameter of the IMS PSB properties. If the GSROLBOK value is NO, IMS issues user abend code 0777, and the application does not receive an SQLCODE. If the GSROLBOK value is YES, the application receives a failure indication on the ESS call followed by an internal IMS ROLB, which results in SQLCODE -911 for DB2.
  - For an MPP, IFP, or message driven BMP, IMS issues user abend code 0777, rolls back all uncommitted changes, and reschedules the transaction. The application does not receive an SQLCODE.

Related information:

PSB segment type format

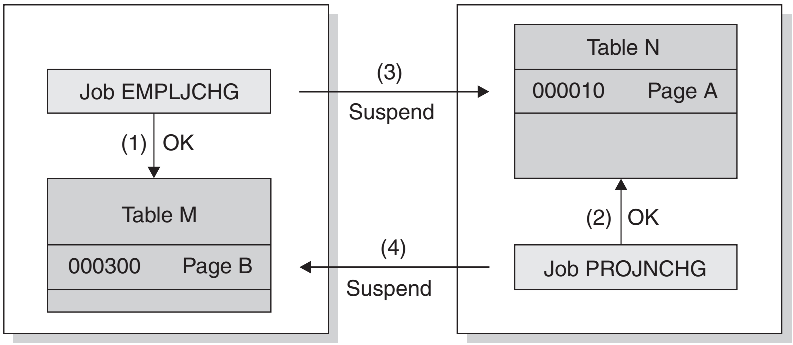
PSBGEN statement

Commit and rollback operations do not timeout. The STOP DATABASE command, however, can time out, in which case DB2 sends messages to the console. When this happens DB2 retries the STOP DATABASE command as many as 15 times.

Deadlock

A *deadlock* occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed. After a preset time interval, DB2 can roll back the current unit of work for one of the processes or request a process to terminate. In determining which process to roll back or terminate, DB2 assesses many characteristics of the processes that are involved in the deadlock and chooses the one that, if terminated, will cause the least impact relative to the other processes. By choosing a process to roll back or terminate, DB2 frees the locks and allows the remaining processes to continue.

The following figure illustrates a deadlock between two transactions.



Notes:

1. Jobs EMPLJCHG and PROJNCHG are two transactions. Job EMPLJCHG accesses table M, and acquires an exclusive lock for page B, which contains record 000300.
2. Job PROJNCHG accesses table N, and acquires an exclusive lock for page A, which contains record 000010.
3. Job EMPLJCHG requests a lock for page A of table N while still holding the lock on page B of table M. The job is suspended, because job PROJNCHG is holding an exclusive lock on page A.
4. Job PROJNCHG requests a lock for page B of table M while still holding the lock on page A of table N. The job is suspended, because job EMPLJCHG is holding an exclusive lock on page B. The situation is a deadlock.

Figure 10. A deadlock example

After a preset time interval (the value of DEADLOCK TIME), DB2 can roll back the current unit of work for one of the processes or request a process to terminate. That frees the locks and allows the remaining processes to continue. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0172. Reason code 00C90088 is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. (The codes that describe the exact DB2 response depend on the operating environment.)

It is possible for two processes to be running on distributed DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

Deadlocks and TSO, Batch, and CAF

When a deadlock or timeout occurs in these environments, DB2 attempts to roll back the SQL for one of the application processes. If the ROLLBACK is successful, that application receives SQLCODE -911. If the ROLLBACK fails, and the application does not abend, the application receives SQLCODE -913.

Deadlocks and IMS

If you are using IMS, and a deadlock occurs, the following actions take place:

- In a DL/I batch application, the application process abnormally terminates with a completion code of 04E and a reason code of 00D44033 or 00D44050.
- In any IMS environment except DL/I batch:
  - DB2 performs a rollback operation on behalf of your application process to undo all DB2 updates that occurred during the current unit of work.
  - For a non-message driven BMP, IMS issues a rollback operation on behalf of your application. If this operation is successful, IMS returns control to your application, and the application receives SQLCODE -911.

If the operation is not successful, the response depends on the value of GSROLBOK parameter of the IMS PSB properties. If the GSROLBOK value is NO, IMS issues user abend code 0777, and the application does not receive an SQLCODE. If the GSROLBOK value is YES, the application receives a failure indication on the ESS call followed by an internal IMS ROLB, which results in SQLCODE -911 for DB2.

- For an MPP, IFP, or message driven BMP, IMS issues user abend code 0777, rolls back all uncommitted changes, and reschedules the transaction. The application does not receive an SQLCODE.

Related information:

PSB segment type format

PSBGEN statement

Deadlocks and CICS

If you are using CICS and a deadlock occurs, the CICS attachment facility decides whether or not to roll back one of the application processes, based on the value of the ROLBE or ROLBI parameter. If your application process is chosen for rollback, it receives one of two SQLCODEs in the SQLCA:

-911 A SYNCPOINT command with the ROLLBACK option was issued on behalf of your application process. All updates (CICS commands and DL/I calls, as well as SQL statements) that occurred during the current unit of work have been undone. (SQLSTATE '40001')

-913 A SYNCPOINT command with the ROLLBACK option was not issued. DB2 rolls back only the incomplete SQL statement that encountered the deadlock or timed out. CICS does not roll back any resources. Your application process should either issue a SYNCPOINT command with the ROLLBACK option itself or terminate. (SQLSTATE '57033')

Consider using the DSNTIAC subroutine to check the SQLCODE and display the SQLCA. Your application must take appropriate actions before resuming. PSPI

Related concepts:

Deadlock detection scenarios

How DB2 calculates the wait time for timeouts

SQL communication area (SQLCA) (DB2 SQL)

Related tasks:

Investigating and resolving timeout situations

Setting installation options for wait times

Related reference:

DEADLOCK TIME field (DB2 Installation and Migration)

ROLLBACK (DB2 SQL)

GET DIAGNOSTICS (DB2 SQL)

Related information:

-911 (DB2 Codes)

-913 (DB2 Codes)

Investigating and resolving timeout situations

Timeout situations can occur for many reasons, including factors relating to both DB2 or IRLM.

Procedure

To investigate and resolve timeout situations:

1. Check the LOCKRULE column value in SYSIBM.SYSTABLESPACE for the table space being accessed. A lock size of TABLE or TABLESPACE might cause a timeout. If your application does not need to lock the entire table or table space, you can resolve the timeout by changing the lock size to PAGE or ANY.
2. Check the number of LOCKS PER TABLE(SPACE), which was set when DB2 was installed. If many page or row locks are acquired and held, a small value for LOCKS PER TABLE(SPACE) might cause lock escalation. You can resolve the timeout by changing the value on the DSNTIPK update panel or by specifying the RELEASE(COMMIT) bind options and committing changes more frequently.
3. Check EXPLAIN output for the failing SQL statement and examine the value of the PLAN_TABLE.TSLOCKMODE column for every table or table space. If the competing applications are attempting to obtain incompatible locks, you might be able to resolve the timeout by running the applications sequentially rather than concurrently. You might also resolve the problem by changing lock size and the following bind options: RELEASE and ISOLATION.
4. Check the RELEASE bind option for the application. Binding with RELEASE(DEALLOCATE) causes partition, table, table space, and DBD locks to be held longer than binding with RELEASE(COMMIT), possibly causing a timeout.
5. Issue the DISPLAY DATABASE command and specify the LOCKS keyword during program execution. By doing so, you can verify that only expected locks are held when the timeout occurs. If unexpected locks are held, you might also resolve the problem by changing lock size and the following bind options: RELEASE and ISOLATION.
6. Check the ISOLATION bind option of the application. The isolation level affects whether locks are acquired and how long they are held.
7. Increase the wait time if the condition is caused by an undetected deadlock. For example, an agent might be holding a lock on the resource for longer than the specified time or the IRLM default wait time.

   If the time limit is too small, you can increase the limit by specifying a new IRLM locked resource wait time limit. You can take one of the following actions to make the change:

   - Use the parameter in the DSN6SPRM assembler macro in the DSNTIJUZ job stream
   - Update the 'wait-time' field of the DB2 installation IMS Resource Lock Manager panel, IRLMRWT.

   You must also specify all other parameters contained in this version of the CSECT. Then take one of the following actions:

   - Resubmit installation job DSNTIJUZ with the link-edit SYSIN file 'NAME' parameter that matches the -START DB2 'PARM=' parameter.
   - Reassemble DSN6SPRM and relink-edit DSNZPARM by resubmitting installation job DSNTIJUZ. The link-edit SYSIN file 'NAME' parameter must match the -START DB2 'PARM=' parameter.

Related tasks:

Using EXPLAIN to identify locks chosen by DB2

Choosing a RELEASE option

Choosing an ISOLATION option

Installation step 5: Define DB2 initialization parameters: DSNTIJUZ (DB2 Installation and Migration)

Related reference:

SYSIBM.SYSTABLESPACE table (DB2 SQL)

LOCKS PER TABLE(SPACE) field (NUMLKTS subsystem parameter) (DB2 Installation and Migration)

RESOURCE TIMEOUT field (IRLMRWT subsystem parameter) (DB2 Installation and Migration)

-DISPLAY DATABASE (DB2) (DB2 Commands)

-START DB2 (DB2) (DB2 Commands)

Related information:

-911 (DB2 Codes)

-913 (DB2 Codes)

DSNT376I (DB2 Messages)

Transaction locks

Understanding the sizes, durations, modes, and objects of transaction locks can help you understand why processes encounter suspension or timeout or deadlock, and how you might prevent that contention.

Related concepts:

Lock contention

Lock size

The *size* (sometimes *scope* or *level*) of a lock on data in a table describes the amount of data that is controlled by the lock. The same piece of data can be controlled by locks of different sizes.

PSPI

DB2 uses locks of the following sizes:

- Table space
- Table
- Partition
- Page
- Row
- LOB
- XML

A table space lock (the largest size) controls the most data, all the data in an entire table space. A page or row lock controls only the data in a single page or row.

As the following figure suggests, row locks and page locks occupy an equal place in the hierarchy of lock sizes.

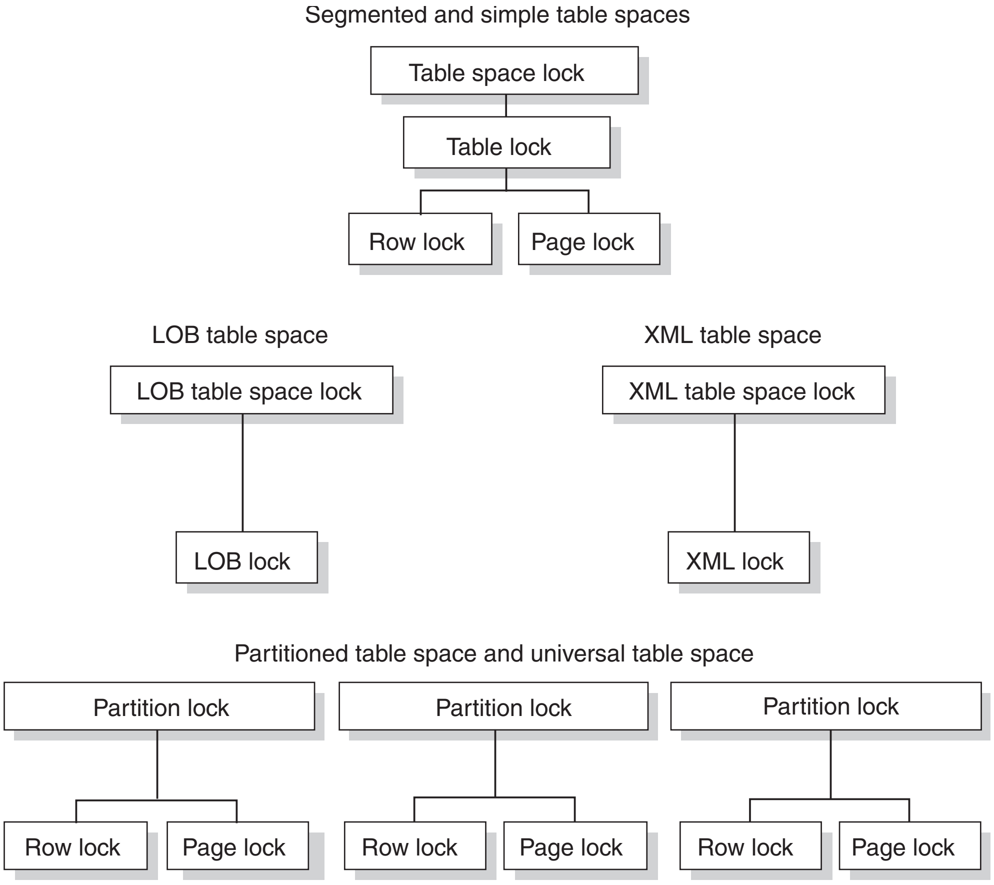


Figure 11. Sizes of objects locked

Locking larger or smaller amounts of data allows you to trade performance for concurrency. Using page or row locks instead of table or table space locks has the following effects:

- Concurrency usually improves, meaning better response times and higher throughput rates for many users.
- Processing time and use of storage increases. That is especially evident in batch processes that scan or update a large number of rows.

Using only table or table space locks has the following effects:

- Processing time and storage usage is reduced.
- Concurrency can be reduced, meaning longer response times for some users but better throughput for one user.

Lock sizes and table space type

DB2 uses different lock sizes depending on the type of table spaces where the locks are acquired.

Partitioned and universal table space
In a partitioned table space or universal table space, locks are obtained at the partition level. Individual partitions are locked when necessary, as they

are accessed. Gross locks (S, U, or X) can be obtained on individual partitions instead of on the entire partitioned table space.

Restriction: If one of the following conditions is true, DB2 must lock all partitions:
- The table space is defined with LOCKSIZE TABLESPACE.
- The LOCK TABLE statement is used without the PART option.

Segmented table space
In a segmented table space without partitions, rows from different tables are contained in different pages. Locking a page does not lock data from more than one table. Also, DB2 can acquire a table lock, which locks only the data from one specific table. Because a single row, of course, contains data from only one table, the effect of a row lock is the same as for a simple or partitioned table space: it locks one row of data from one table.

Simple table space
DB2 no longer supports the creation of simple table spaces. However, an existing simple table space can contain more than one table. A lock on the table space locks all the data in every table. A single page of the table space can contain rows from every table. A lock on a page locks every row in the page, no matter what tables the data belongs to. Thus, a lock needed to access data from one table can make data from other tables temporarily unavailable. That effect can be partly undone by using row locks instead of page locks.

LOB table space
In a LOB table space, pages are not locked. Because the concept of rows does not occur in a LOB table space, rows are not locked. Instead, LOBs are locked.

XML table space
In an XML table space, XML locks are acquired.

Example: simple versus segmented table spaces

Suppose that tables T1 and T2 reside in table space TS1. In a simple table space, a single page can contain rows from both T1 and T2. If User 1 and User 2 acquire incompatible locks on different pages, such as exclusive locks for updating data, neither can access all the rows in T1 and T2 until one of the locks is released. (User 1 and User 2 can both hold a page lock on the same page when the mode of the locks are compatible, such as locks for reading data.)

As the figure also shows, in a segmented table space, a table lock applies only to segments assigned to a single table. Thus, User 1 can lock all pages assigned to the segments of T1 while User 2 locks all pages assigned to segments of T2. Similarly, User 1 can lock a page of T1 without locking any data in T2.

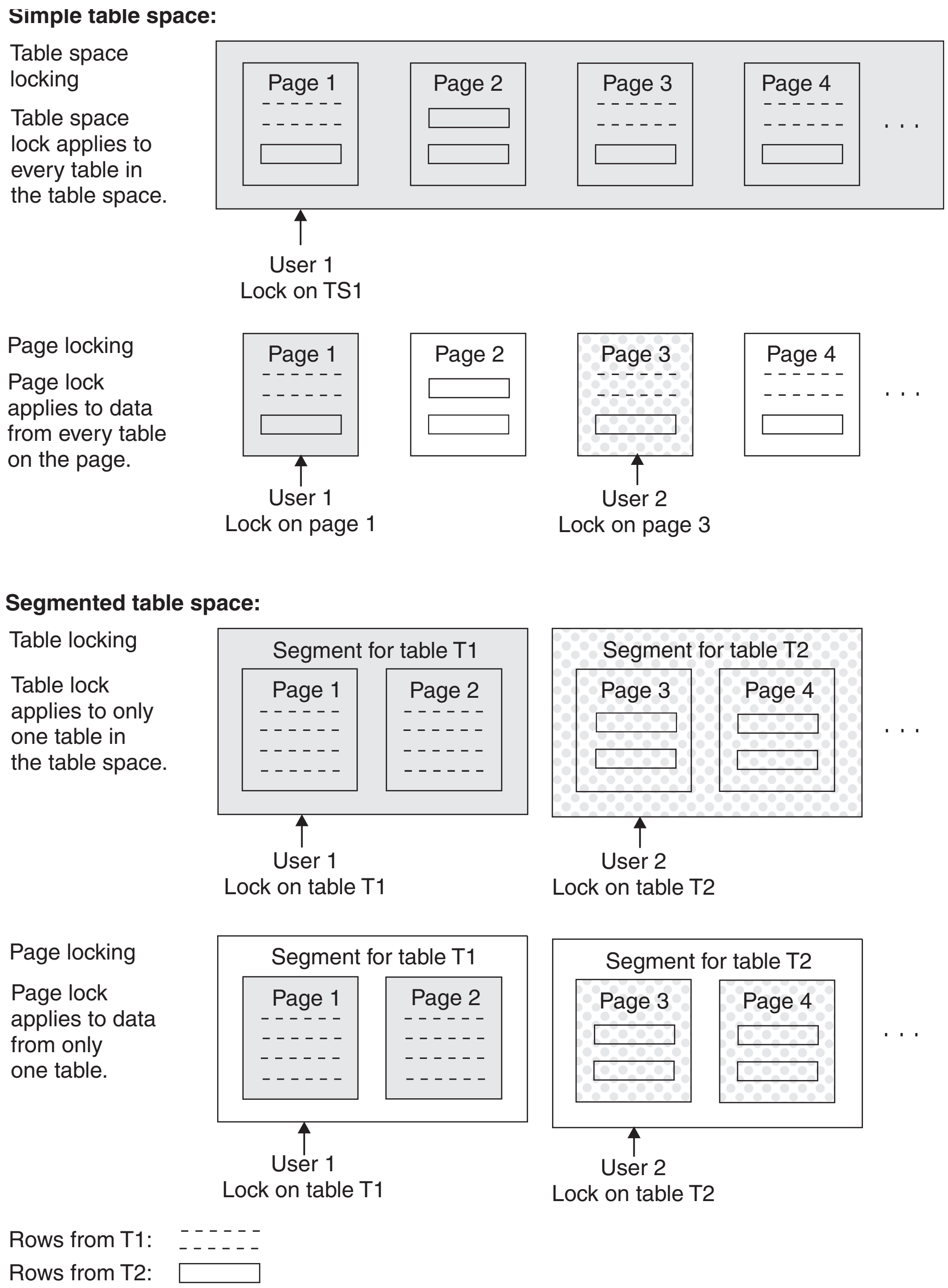


Figure 12. Page locking for simple and segmented table spaces

PSPI

Related concepts:

Types of DB2 table spaces (Introduction to DB2 for z/OS)

Related tasks:

Specifying the size of locks for a table space

Controlling lock size for LOB table spaces

Specifying the size of locks for XML data

Related reference:

Locks acquired for SQL statements

LOCK TABLE (DB2 SQL)

The duration of a lock

The *duration* of a lock is the length of time the lock is held. It varies according to when the lock is acquired and when it is released.

For maximum concurrency, locks on a small amount of data held for a short duration are better than locks on large amounts of data that are held for long durations. However, acquiring locks requires processor time, and holding locks requires storage. Therefore, acquiring and holding one table space lock is more efficient than acquiring and holding many page locks. You must consider that trade-off to meet your performance and concurrency objectives.

Partition, table, and table space locks

These locks are acquired when the application first accesses the object. The RELEASE bind option controls when the locks are released. Locks can be released at the next commit point or be held until the program terminates.

When statements from more than one package acquire partition locks for the same table space, all partition locks are held for the same duration. If the first statement to access the table space is from a package that uses the RELEASE(COMMIT) bind option, all partition locks follow the rules of RELEASE(COMMIT). However, if a statement from another package that uses the RELEASE(DEALLOCATE) bind option accesses a partition in the same tables space, all partition locks are then promoted to follow the rules of RELEASE(DEALLOCATE).

The locks can also be held past commit points for cursors that are defined with the WITH HOLD option.

Page and row locks

If a page or row is locked, DB2 acquires the lock only when it is needed. When the lock is released depends on many factors, but the lock is rarely held beyond the next commit point.

You can use the following bind options to take some control over the duration of locks:

- ISOLATION
- RELEASE
- CURRENTDATA

Related concepts:

XML lock and XML table space lock duration

LOB lock and LOB table space lock duration

Held and non-held cursors (DB2 Application programming and SQL)

Related tasks:

Choosing a RELEASE option

Choosing an ISOLATION option

Choosing a CURRENTDATA option

Related reference:

COMMIT (DB2 SQL)

Lock modes

The *mode* of a lock tells what access to the locked object is permitted to the lock owner and to any concurrent processes.

PSPI

DB2 uses of the lock modes to determine whether one lock is *compatible* with another. Some lock modes do not exclude all other users. For example, assume that application process A holds a lock on a table space that process B also wants to access. DB2 requests, on behalf of process B, a lock of some particular mode. If the mode of the lock for process A permits the lock requested by process B, the modes of the two locks are said to be compatible. However, if the two locks are not compatible, process B cannot proceed. It must wait until process A releases its lock, and until all other existing incompatible locks are released.

Page and row lock modes

The modes and their effects are listed in the order of increasing control over resources.

S lock (share)

The lock owner and any concurrent processes can read, but not change, the locked page or row. Concurrent processes can acquire S or U locks on the page or row or might read data without acquiring a page or row lock

U lock (update)

The lock owner can read, but not change, the locked page or row. Concurrent processes can acquire S-locks or might read data without acquiring a page or row lock, but no concurrent process can acquire a U-lock.

U locks reduce the chance of deadlocks when the lock owner is reading a page or row to determine whether to change it. The owner can start with the U lock and then promote the lock to an X lock to change the page or row.

X lock (exclusive)

The lock owner can read or change the locked page or row. A concurrent process cannot acquire S, U, or X locks on the page or row. However, concurrent processes, such as those processes bound with the CURRENTDATA(NO) or ISOLATION(UR) bind options or running with YES specified for the EVALUNC subsystem parameter, can read the data without acquiring a page or row lock.

Partition, table space, and table lock modes

When a page or row is locked, the table, partition, or table space that contains it is also locked. This *intent lock* indicates the plan that the application process has for accessing the data. In that case, the table, partition, or table space lock has one of the *intent modes*: either IS for *intent share*, IX for *intent exclusive*, or SIX for *share with intent exclusive*.

The modes S, U, and X of table, partition, and table space locks are sometimes called *gross modes*. In the context of reading, SIX is a gross mode lock because you do not get page or row locks; in this sense, it is like an S lock.

The modes and their effects are listed in the order of increasing control over resources.

IS lock (intent share)
The lock owner can read data in the table, partition, or table space, but not change it. Concurrent processes can both read and change the data. The lock owner might acquire a page or row lock on any data it reads.

IX lock (intent exclusive)
The lock owner and concurrent processes can read and change data in the table, partition, or table space. The lock owner might acquire a page or row lock on any data it reads; it must acquire one on any data it changes.

S lock (share)
The lock owner and any concurrent processes can read, but not change, data in the table, partition, or table space. The lock owner does not need page or row locks on data it reads.

U lock (update)
The lock owner can read, but not change, the locked data; however, the owner can promote the lock to an X lock and then can change the data. Processes concurrent with the U lock can acquire S locks and read the data, but no concurrent process can acquire a U lock. The lock owner does not need page or row locks.

U locks reduce the chance of deadlocks when the lock owner is reading data to determine whether to change it. U locks are acquired on a table space when the lock size is TABLESPACE and the statement is a SELECT with a FOR UPDATE clause. Similarly, U locks are acquired on a table when lock size is TABLE and the statement is a SELECT with a FOR UPDATE clause.

SIX lock (share with intent exclusive)
The lock owner can read and change data in the table, partition, or table space. Concurrent processes can read data in the table, partition, or table space, but not change it. Only when the lock owner changes data does it acquire page or row locks.

X lock (exclusive)
The lock owner can read or change data in the table, partition, or table space. A concurrent process can access the data if the process runs with UR isolation or if data in a partitioned table space is running with CS isolation and CURRENTDATA((NO). The lock owner does not need page or row locks.

Example: An SQL statement locates John Smith in a table of customer data and changes his address. The statement locks the entire table space in mode IX and the specific row that it changes in mode X.

Compatibility of lock modes

The following shows whether page locks of any two modes, or row locks of any two modes are compatible. No question of compatibility arises between page and row locks, because a table space cannot use both page and row locks.

Table 39. Compatibility matrix of page lock and row lock modes

Lock mode	Share (S-lock)	Update (U-lock)	Exclusive (X-lock)
Share (S-lock)	Yes	Yes	No

Table 39. Compatibility matrix of page lock and row lock modes (continued)

Lock mode	Share (S-lock)	Update (U-lock)	Exclusive (X-lock)
Update (U-lock)	Yes	No	No
Exclusive (X-lock)	No	No	No

Compatibility for table space locks is slightly more complex that for page and row locks. The following table shows whether table space locks of any two modes are compatible.

Table 40. Compatibility of table and table space (or partition) lock modes

Lock Mode	IS	IX	S	U	SIX	X
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No	No
S	Yes	No	Yes	Yes	No	No
U	Yes	No	Yes	No	No	No
SIX	Yes	No	No	No	No	No
X	No	No	No	No	No	No

PSPI

Related reference:

LOB and LOB table space lock modes

“XML and XML table space lock modes” on page 218

How access paths affect locks

The access path that DB2 uses can affect the mode, size, and even the object of a lock.

For example, an UPDATE statement using a table space scan might need an X lock on the entire table space. If rows to be updated are located through an index, the same statement might need only an IX lock on the table space and X locks on individual pages or rows.

If you use the EXPLAIN statement to investigate the access path chosen for an SQL statement, then check the lock mode in column TSLOCKMODE of the resulting PLAN_TABLE. If the table resides in a nonsegmented or universal table space, or is defined with LOCKSIZE TABLESPACE, the mode shown is that of the table space or partition lock. Otherwise, the mode is that of the table lock.

Important points that you should consider when you work with DB2 locks include:

- You usually do not have to lock data explicitly in your program.
- DB2 ensures that your program does not retrieve uncommitted data unless you specifically allow that.
- Any page or row where your program updates, inserts, or deletes stays locked at least until the end of a unit of work, regardless of the isolation level. No other process can access the object in any way until then, unless you specifically allow that access to that process.
- Commit often for concurrency. Determine points in your program where changed data is consistent. At those points, you should issue:

TSO, Batch, and CAF
An SQL COMMIT statement

IMS A CHKP or SYNC call, or (for single-mode transactions) a GU call to the I/O PCB

CICS A SYNCPOINT command.

- Set ISOLATION (usually RR, RS, or CS) when you bind the plan or package.
  - With RR (repeatable read), all accessed pages or rows are locked until the next commit point.
  - With RS (read stability), all qualifying pages or rows are locked until the next commit point.
  - With CS (cursor stability), only the pages or rows currently accessed can be locked, and those locks might be avoided. (You can access one page or row for each open cursor.)
- You can also use an *isolation clause* to specify the isolation for specific SQL statements.
- A deadlock can occur if two processes each hold a resource that the other needs. One process is chosen as “victim”, its unit of work is rolled back, and an SQL error code is issued.
-

  GUPI

  You can lock an entire nonsegmented table space, or an entire table in a segmented table space, by the LOCK TABLE statement:
  - To let other users retrieve, but not update, delete, or insert, issue the following statement:

    ```
    LOCK TABLE table-name IN SHARE MODE
    ```

  - To prevent other users from accessing rows in any way, except by using UR isolation, issue the following statement:

    ```
    LOCK TABLE table-name IN EXCLUSIVE MODE
    ```

  GUPI

Related tasks:

Designing databases for concurrency

Choosing an ISOLATION option

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

LOCK TABLE (DB2 SQL)

COMMIT (DB2 SQL)

Objects that are subject to locks

DB2 uses locks on various types of user and system object types to control changes to data by concurrent operations.

PSPI

DB2 uses locks on the following types of objects:

User data in target tables

A *target table* is a table that is accessed specifically in an SQL statement, and especially one that the statement updates, either by name or through a view. Locks on those tables are the most common concern, and the ones over which you have most control.

User data in related tables

Operations that are subject to referential constraints can require locks on related tables. For example, if you delete from a parent table, DB2 might delete rows from the dependent table as well. In that case, DB2 locks data in the dependent table as well as in the parent table.

Similarly, operations on rows that contain LOB or XML values might require locks on the LOB or XML table space and possibly on LOB or XML values within that table space. For more information, see Locks for LOB data and Locks for XML data.

If your application uses triggers, any triggered SQL statements can cause additional locks to be acquired.

Indexes and data-only locking

Instead of acquiring locks on index pages, DB2 uses a technique called *data-only locking* to serialize changes. However, in data sharing environments, DB2 uses index page p-locks.

Index page latches are acquired to serialize changes within a page and guarantee that the page is physically consistent. Acquiring page latches ensures that transactions accessing the same index page concurrently do not see the page in a partially changed state.

The underlying data page or row locks are acquired to serialize the reading and updating of index entries to ensure the data is logically consistent, meaning that the data is committed and not subject to rollback or abort. The data locks can be held for a long duration such as until commit. However, the page latches are only held for a short duration while the transaction is accessing the page. Because the index pages are not locked, hot spot insert scenarios (which involve several transactions trying to insert different entries into the same index page at the same time) do not cause contention problems in the index.

A query that uses index-only access might lock the data page or row, and that lock can contend with other processes that lock the data. However, using lock avoidance techniques can reduce the contention.

Pseudo-deleted index entries and data-locking

When data rows are deleted, index entries are not physically deleted unless the delete operation has exclusive control over the index page set. These index entries are called *pseudo-deleted index entries*. Subsequent searches continue to access these pseudo-deleted entries, which can gradually degrade performance as more rows are deleted. These pseudo-deleted index entries can also result in timeouts and deadlocks for applications that insert data into tables with unique indexes. *Pseudo-empty index pages* are pages that contain only pseudo-deleted index entries. The REORG utility removes pseudo-deleted index entries and pseudo-empty index pages when you run it to reorganize the data.

Data-only locking for XML data

When DB2 searches using XML values (the first key values) in the XML index key entries, it does not acquire the index page latch and does not lock either the base table data pages or rows, or the XML table space. When the matched-value index key entries are found, the corresponding DOCID values (the second key value) are retrieved. The retrieved DOCID values are used to retrieve the base table RIDs using the DOCID index. The regular data-only locking technique is applied on the DOCID index page and base table data page or row.

DB2 catalog objects

SQL data definition statements, GRANT statements, and REVOKE statements require locks on the DB2 catalog. If different application processes are issuing these types of statements, catalog contention can occur.

SQL statements that update the catalog table space SYSDBASE contend with each other when those statements are on the same table space. Those statements are:

- CREATE TABLESPACE, TABLE, and INDEX
- ALTER TABLESPACE, TABLE, INDEX
- DROP TABLESPACE, TABLE, and INDEX
- CREATE VIEW, SYNONYM, and ALIAS
- DROP VIEW and SYNONYM, and ALIAS
- COMMENT ON and LABEL ON
- GRANT and REVOKE of table privileges
- RENAME TABLE
- RENAME INDEX
- ALTER VIEW

Recommendations:

- Reduce the concurrent use of statements that update SYSDBASE for the same table space.
- When you alter a table or table space, quiesce other work on that object.

The following contention situations are independent of the referenced database:

- CREATE and DROP statements for a table space or index that uses a storage group contend significantly with other such statements.
- CREATE, ALTER, and DROP DATABASE, and GRANT and REVOKE database privileges all contend with each other and with any other function that requires a database privilege.
- CREATE, ALTER, and DROP STOGROUP contend with any SQL statements that refer to a storage group and with extensions to table spaces and indexes that use a storage group.
- GRANT and REVOKE for plan, package, system, or use privileges contend with other GRANT and REVOKE statements for the same type of privilege and with data definition statements that require the same type of privilege.

PSPI

Skeleton cursor tables (SKCT) for application plans and skeleton package tables (SKPT) for packages

The following operations require incompatible locks on the SKCT or SKPT, whichever is applicable, and cannot run concurrently:

- Binding, rebinding, or freeing the plan or package
- Dropping a resource or revoking a privilege that the plan or package depends on
- In some cases, altering a resource that the plan or package depends on

Database descriptors (DBDs) that represent databases

If the DBD is not in the EDM DBD cache, most processes acquire locks on the database descriptor table space (DBD01), which has the effect of locking the DBD and can cause conflict with other processes. However, if he DBD is in the EDM DBD cache, the lock on the DBD depends on the type of process, as shown in the following table:

Table 41. Contention for locks on a DBD in the EDM DBD cache

Process Type	Process	Lock acquired	Conflicts with process type
1	Static SQL data manipulation statements (such as SELECT, INSERT, UPDATE, DELETE)[1]	none[3]	none[3]
2	Dynamic SQL data manipulation SQL statements[2]	S	3
3	Data definition statements (ALTER, CREATE, DROP)	X	2,3,4
4	Utilities	S	3

Notes:

1. Static SQL statements can conflict with other processes because of locks on data.
2. If caching of dynamic SQL is turned on, no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache.
3. When referential integrity is involved, S-locks are held on the DBD to ensure serialization. These locks can conflict with data definition statements.

PSPI

Related concepts:

EDM storage

DB2 directory (Introduction to DB2 for z/OS)

Lock avoidance

Page set P-Locks (DB2 Data Sharing Planning and Administration)

Related tasks:

Avoiding locks during predicate evaluation

Improving concurrency for applications that tolerate incomplete results

Maintaining data organization

Related reference:

Locks acquired for SQL statements

Modes of transaction locks for various processes

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

Avoiding catalog contention when dropping a table space

Dropping an object can cause catalog contention because the DB2 database manager must remove all rows that relate to that object from all tables in the catalog and directory. When the database manager removes the catalog and directory table rows, it must get an X lock on every row.

Procedure

Take the following steps to avoid catalog contention when you explicitly or implicitly drop a table space. These steps reduce the number of locks that the database manager must acquire when it processes the DROP statement.

1. Run the MODIFY STATISTICS utility with the DELETE and AGE(*) options on the table space to remove all statistics history for the table space.
2. Run the MODIFY RECOVERY utility on the table space to remove rows from the SYSIBM.SYSCOPY catalog table, the SYSIBM.SYSOBDS catalog table, and the SYSIBM.SYSLGRNX directory table that are related to the table space.
3. Drop the table space.
4. Issue the COMMIT statement as soon as possible after you drop the table space.

Related reference:

MODIFY STATISTICS (DB2 Utilities)

MODIFY RECOVERY (DB2 Utilities)

How DB2 chooses lock types

Different types of SQL data manipulation statements acquire locks on target tables.

PSPI

The lock acquired because of an SQL statement is not always a constant throughout the time of execution. In certain situations,DB2 can change acquired locks during execution. Many other processes and operations acquire locks.

PSPI

Locks acquired for SQL statements

When SQL statements access or modify data, locks must be acquired to prevent other applications from accessing data that has been changed but not committed. How and when the locks are acquired for a particular SQL statement depend on the type of processing, the access method, and attributes of the application and target tables.

PSPI

The following tables show the locks that certain SQL processes acquire and the modes of those locks. Whether locks are acquired at all and the mode of those locks depend on the following factors:

- The type of processing being performed
- The value of LOCKSIZE for the target table

- The isolation level of the plan, package, or statement
- The method of access to data
- Whether the application uses the SKIP LOCKED DATA option

Example SQL statement

The following SQL statement and sample steps provide a way to understand the following tables.

```
EXEC SQL DELETE FROM DSN8A10.EMP WHERE CURRENT OF C1;
```

Use the following sample steps to understand the table:

1. Find the portion of the table that describes DELETE operations using a cursor.
2. Find the row for the appropriate values of LOCKSIZE and ISOLATION. Table space DSN8A10 is defined with LOCKSIZE ANY. The default value of ISOLATION is CS with CURRENTDATA (NO) by default.
3. Find the sub-row for the expected access method. The operation probably uses the index on employee number. Because the operation deletes a row, it must update the index. Hence, you can read the locks acquired in the sub-row for "Index, updated":
   - An IX lock on the table space
   - An IX lock on the table (but see the step that follows)
   - An X lock on the page containing the row that is deleted
4. Check the notes to the entries you use, at the end of the table. For this sample operation, see:
   - Note 2, on the column heading for "Table". If the table is not segmented, or if the table is segmented and partitioned, no separate lock is taken on the table.
   - Note 3, on the column heading for "Data Page or Row". Because LOCKSIZE for the table space is ANY, DB2 can choose whether to use page locks, table locks, or table space locks. Typically it chooses page locks.

SELECT with read-only cursor, ambiguous cursor, or no cursor

The following table shows locks that are acquired during the processing of SELECT with read-only or ambiguous cursor, or with no cursor SQL statements. UR isolation is allowed and requires none of these locks.

Table 42. Locks acquired for SQL statements SELECT with read-only or ambiguous cursor

LOCKSIZE	ISOLATION	Access method[1]	Lock Mode Table space	Lock Mode Table	Lock Mode Data Page or Row
TABLESPACE	CS RS RR	Any	S	n/a	n/a
TABLE[2]	CS RS RR	Any	IS	S	n/a
PAGE, ROW, or ANY	CS	Index, any use	IS[4, 10]	IS[4]	S[5]
		Table space scan	IS[4, 10]	IS[4]	S[5]
PAGE, ROW, or ANY	RS	Index, any use	IS[4, 10]	IS[4, 11] or IX	S[5], U[11], or X[11]
		Table space scan	IS[4, 10, 11] or IX	IS[4, 11]	S[5], U[11], or X[11]
PAGE, ROW, or ANY	RR	Index/data probe	IS[4, 11]	IS[4, 11]	S[5], U[11], or X[11]
		Index scan[6]	IS[4, 11], IX, or S	S, IS[4, 11], IX, or n/a	S[5], U[11], X[11], or n/a
		Table space scan[6]	IS[2]or S	S or n/a	n/a

INSERT, VALUES(...), or INSERT fullselect[7]

The following table shows locks that are acquired during the processing of INSERT, VALUES(...), or INSERT fullselect SQL statements.

Table 43. Locks acquired for SQL statements INSERT ... VALUES(...) or INSERT ... fullselect

LOCKSIZE	ISOLATION	Access method[1]	Lock Mode Table space[9]	Lock Mode Table[2]	Lock Mode Data Page or Row[3]
TABLESPACE	CS RS RR	Any	X	n/a	n/a
TABLE[2]	CS RS RR	Any	IX	X	n/a
PAGE, ROW, or ANY	CS RS RR	Any	IX	IX	X

UPDATE or DELETE without cursor

The following table shows locks that are acquired during the processing of UPDATE or DELETE without curso SQL statements. Data page and row locks apply only to selected data.

Table 44. Locks acquired for SQL statements UPDATE, or DELETE without cursor

LOCKSIZE	ISOLATION	Access method[1]	Lock Mode Table space[9]	Lock Mode Table[2]	Lock Mode Data Page or Row[3]
TABLESPACE	CS RS RR	Any	X	n/a	n/a
TABLE[2]	CS RS RR	Any	IX	X	n/a
PAGE, ROW, or ANY	CS	Index selection	IX	IX	• For delete: X • For update: U [8]→X
		Index/data selection	IX	IX	U[8]→X
		Table space scan	IX	IX	U[8]→X
PAGE, ROW, or ANY	RS	Index selection	IX	IX	• For update: S or U[8, 11, 12] →X • For delete: [S→X][8] or X
		Index/data selection	IX	IX	S or U[8, 11, 12] →X
		Table space scan	IX	IX	S or U[8, 11, 12] →X
PAGE, ROW, or ANY	RR	Index selection	IX	IX	• For update: [S or U→X][8, 11, 12] or X • For delete: [S→X] or X
		Index/data selection	IX	IX	S or U[8, 11, 12] →X
		Table space scan	IX[2] or X	X or n/a	n/a

SELECT with FOR UPDATE OF

The following table shows locks that are acquired during the processing of SELECT with FOR UPDATE OF SQL statements. Data page and row locks apply only to selected data.

Table 45. Locks acquired for SQL statements SELECT with FOR UPDATE OF

LOCKSIZE	ISOLATION	Access method[1]	Lock Mode Table space[9]	Lock Mode Table[2]	Lock Mode Data Page or Row[3]
TABLESPACE	CS RS RR	Any	S or U[12]	n/a	n/a
TABLE[2]	CS RS RR	Any	IS or IX	U	n/a
PAGE, ROW, or ANY	CS	Index, any use	IX	IX	U
		Table space scan	IX	IX	U
PAGE, ROW, or ANY	RS	Index, any use	IX	IX	S, U, or X[11, 12]
		Table space scan	IX	IX	S, U, or X[11, 12]
PAGE, ROW, or ANY	RR	Index/data probe	IX	IX	S, U, or X[11, 12]
		Index scan[6]	IX or X	X, IX, or n/a	S, U, X[11, 12], or n/a
		Table space scan[6]	IX[2] or X	X or n/a	S, U, X[11, 12], or n/a

UPDATE or DELETE with cursor

The following table shows locks that are acquired during the processing of xxx SQL statements.

Table 46. Locks acquired for SQL statements UPDATE or DELETE with cursor

LOCKSIZE	ISOLATION	Access method[1]	Lock Mode Table space[9]	Lock Mode Table[2]	Lock Mode Data Page or Row[3]
TABLESPACE	Any	Any	X	n/a	n/a
TABLE[2]	Any	Any	IX	X	n/a
PAGE, ROW, or ANY	CS, RS, or RR	Index, updated	IX	IX	X
		Index not updated	IX	IX	X

Mass delete or TRUNCATE

Lock modes for TRUNCATE depend solely on the type of tables space regardless of LOCKSIZE or isolation level:

Simple table space
Locks the table space with an X lock

Segmented table space (not partitioned)
Locks the table with an X lock and lock the table space with an IX lock

Partitioned table space (including segmented)
Locks each partition with an X lock

Notes for this topic

1. All access methods are either scan-based or probe-based. Scan-based means the index or table space is scanned for successive entries or rows. Probe-based means the index is searched for an entry as opposed to a range of entries, which a scan does. ROWIDs provide data probes to look for a single data row directly. The type of lock used depends on the backup access method. Access methods might be index-only, data-only, or index-to-data.

   Index-only
   The index alone identifies qualifying rows and the return data.

Data-only:
The data alone identifies qualifying rows and the return data, such as a table space scan or the use of ROWID for a probe.

Index-to-data
The index is used or the index plus data are used to evaluate the predicate:

Index selection
The index is used to evaluate predicate and data is used to return values.

Index/data selection
The index and data are used to evaluate predicate and data is used to return values.

2. Used only for segmented table spaces that are not partitioned.
3. These locks are taken on pages if LOCKSIZE is PAGE or on rows if LOCKSIZE is ROW. When the maximum number of locks per table space (LOCKMAX) is reached, locks escalate to a table lock for tables in a segmented table space without partitions, or to a table space lock for tables in a non-segmented table space. Using LOCKMAX 0 in CREATE or ALTER TABLESPACE disables lock escalation.
4. If the table or table space is started for read-only access, DB2 attempts to acquire an S lock. If an incompatible lock already exists, DB2 acquires the IS lock.
5. SELECT statements that do not use a cursor, or that use read-only or ambiguous cursors and are bound with CURRENTDATA(NO), might not require any lock if DB2 can determine that the data to be read is committed. This is known as *lock avoidance*. If your application can tolerate incomplete or inconsistent results, you can also specify the SKIP LOCKED DATA option in your query to avoid lock wait times.
6. Even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE. If that occurs, SQLCODE +806 is issued.
7. The locks listed are acquired on the object into which the insert is made. A subselect acquires additional locks on the objects it reads, as if for SELECT with read-only cursor or ambiguous cursor, or with no cursor.
8. When the value of the XLKUPDLT subsystem parameter is YES, the initial lock is an X-lock.
9. Includes partition locks, and does not include LOB table space locks.
10. If the table space is partitioned, locks can be avoided on the partitions.
11. If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP UPDATE LOCKS option to obtain and hold a U-lock instead of an S-lock, or you can use the USE AND KEEP EXCLUSIVE LOCKS option to obtain and hold an X-lock instead of an S-lock. If page or row locks are used, the table space and table locks are IX-locks. If page and row locks are not used, because a gross lock is acquired on the table or table space, the lock is a U-lock for USE AND KEEP UPDATE LOCKS or an X-lock for USE AND KEEP EXCLUSIVE locks.
12. The RRULOCK subsystem parameter controls the type of lock that is acquired for isolation levels RS and RR. If RRULOCK=YES, a U-lock is acquired. Otherwise, an S-lock is acquired.

PSPI

Related concepts:

Lock size

Locks for LOB data

Lock avoidance

Related tasks:

Specifying the maximum number of locks that a process can hold on a table space

Improving concurrency for applications that tolerate incomplete results

Improving concurrency for update and delete operations

Related reference:

X LOCK FOR SEARCHED U/D field (XLKUPDLT subsystem parameter) (DB2 Installation and Migration)

U LOCK FOR RR/RS field (RRULOCK subsystem parameter) (DB2 Installation and Migration)

SKIP LOCKED DATA (DB2 SQL)

Lock promotion

Lock promotion is the action of exchanging one lock on a resource for a more restrictive lock on the same resource, held by the same application process.

PSPI

Example

An application reads data, which requires an IS lock on a table space. Based on further calculation, the application updates the same data, which requires an IX lock on the table space. The application is said to *promote* the table space lock from mode IS to mode IX.

Effects

When promoting the lock, DB2 first waits until any incompatible locks held by other processes are released. When locks are promoted, they are promoted in the direction of increasing control over resources: from IS to IX, S, or X; from IX to SIX or X; from S to X; from U to X; and from SIX to X.

PSPI

Lock escalation

Lock escalation is the act of releasing a large number of page, row, LOB, or XML locks, held by an application process on a single table or table space, to acquire a table or table space lock, or a set of partition locks, of mode S or X instead.

PSPI

When locks escalation occurs, DB2 issues message DSNI031I, which identifies the table space for which lock escalation occurred, and some information to help you identify what plan or package was running when the escalation occurred.

Lock counts are always kept on a table or table space level. For an application process that is accessing LOBs or XML, the LOB or XML lock count on the LOB or

XML table space is maintained separately from the base table space, and lock escalation occurs separately from the base table space.

When escalation occurs for a partitioned table space, only partitions that are currently locked are escalated. Partitions that have not yet been locked are not affected by lock escalation. Unlocked partitions remain unlocked. After lock escalation occurs, any unlocked partitions that are subsequently accessed are locked with gross locks.

For an application process that is using Sysplex query parallelism, the lock count is maintained on a member basis, not globally across the group for the process. So, escalation on a table space or table by one member does not cause escalation on other members.

Example lock escalation

Assume that a segmented table space without partitions is defined with LOCKSIZE ANY and LOCKMAX 2000. DB2 can use page locks for a process that accesses a table in the table space and can escalate those locks. If the process attempts to lock more than 2000 pages in the table at one time, DB2 promotes its intent lock on the table to mode S or X and then releases its page locks.

If the process is using Sysplex query parallelism and a table space that it accesses has a LOCKMAX value of 2000, lock escalation occurs for a member only if more than 2000 locks are acquired for that member.

When lock escalation occurs

Lock escalation balances concurrency with performance by using page or row locks while a process accesses relatively few pages or rows, and then changing to table space, table, or partition locks when the process accesses many. When it occurs, lock escalation varies by table space, depending on the values of LOCKSIZE and LOCKMAX. Lock escalation is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.

Recommendations

The DB2 statistics and performance traces can tell you how often lock escalation has occurred and whether it has caused timeouts or deadlocks. As a rough estimate, if one quarter of your lock escalations cause timeouts or deadlocks, then escalation is not effective for you. You might alter the table to increase LOCKMAX and thus decrease the number of escalations.

Alternatively, if lock escalation is a problem, use LOCKMAX 0 to disable lock escalation.

Example

Assume that a table space is used by transactions that require high concurrency and that a batch job updates almost every page in the table space. For high concurrency, you should probably create the table space with LOCKSIZE PAGE and make the batch job commit every few seconds.

LOCKSIZE ANY

LOCKSIZE ANY is a possible choice, if you take other steps to avoid lock escalation. If you use LOCKSIZE ANY, specify a LOCKMAX value large enough so that locks held by transactions are not normally escalated. Also, LOCKS PER USER must be large enough so that transactions do not reach that limit.

If the batch job is:

Concurrent with transactions
It must use page or row locks and commit frequently: for example, every 100 updates. Review LOCKS PER USER to avoid exceeding the limit. The page or row locking uses significant processing time. Binding with ISOLATION(CS) might discourage lock escalation to an X table space lock for those applications that read a lot and update occasionally. However, this might not prevent lock escalation for those applications that are update intensive.

Non-concurrent with transactions
It need not use page or row locks. The application could explicitly lock the table in exclusive mode.

PSPI

Related concepts:

The ISOLATION (CS) option

Related tasks:

Specifying the size of locks for a table space

Specifying the maximum number of locks that a process can hold on a table space

Controlling XML lock escalation

Related reference:

LOCKS PER USER field (NUMLKUS subsystem parameter) (DB2 Installation and Migration)

Related information:

DSNI031I (DB2 Messages)

Modes of transaction locks for various processes

DB2 uses different lock modes for different types of processes.

PSPI

The rows in the following table show a sample of several types of DB2 processes. The columns show the most restrictive mode of locks used for different objects and the possible conflicts between application processes.

Table 47. Modes of DB2 transaction locks

Process	Catalog table spaces	Skeleton tables (SKCT and SKPT)	Database descriptor (DBD)[1]	Target table space [2]
Transaction with static SQL	IS [3]	S	n/a [4,5]	Any [6]
Query with dynamic SQL	IS [7]	S	S	Any [6]
BIND process	IX	X	S	n/a

Table 47. Modes of DB2 transaction locks (continued)

Process	Catalog table spaces	Skeleton tables (SKCT and SKPT)	Database descriptor (DBD)[1]	Target table space [2]
SQL CREATE TABLE statement	IX	n/a	X	X[9]
SQL ALTER TABLE statement	IX	X [8]	X	n/a
SQL ALTER TABLESPACE statement	IX	X [11]	X	n/a
SQL DROP TABLESPACE statement	IX	X [10]	X	n/a
SQL GRANT statement	IX	n/a	n/a	n/a
SQL REVOKE statement	IX	X [10]	n/a	n/a

Notes:

1. In a lock trace, these locks usually appear as locks on the DBD.
2. The target table space is one of the following table spaces:
   - Accessed and locked by an application process
   - Processed by a utility
   - Designated in the data definition statement
3. The lock is held briefly to check EXECUTE authority.
4. If the required DBD is not already in the EDM DBD cache, locks are acquired on table space DBD01, which effectively locks the DBD.
5. When referential integrity is involved, an S-lock is held on the DBD to ensure serialization.
6. For detailed information, see “Locks acquired for SQL statements” on page 205.
7. Except while checking EXECUTE authority, IS locks on catalog tables are held until a commit point.
8. The package that uses the SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is added or changed for a table.
9. For segmented table spaces that are not partitioned, an X-lock is acquired on the table space. For universal table spaces, a lock is acquired on partition 1 of the table space.
10. The package using the SKCT or SKPT is marked invalid as a result of this operation.
11. These locks are not held when ALTER TABLESPACE is changing the following options: PRIQTY, SECQTY, PCTFREE, FREEPAGE, CLOSE, and ERASE.

PSPI

Related concepts:

Lock modes

Related tasks:

Using EXPLAIN to identify locks chosen by DB2

Related reference:

ALTER TABLESPACE (DB2 SQL)

Locks for LOB data

The purpose of LOB locks is different from the purpose of regular transaction locks. A lock that is taken on a LOB value in a LOB table space is called a *LOB lock*.

Introductory concepts

Large objects (LOBs) (DB2 SQL)

Large object data types (Introduction to DB2 for z/OS)

Large object table spaces (Introduction to DB2 for z/OS)

PSPI

LOB data values might be stored separately from table space that contains the values in the base table. The separate table space that contains LOB values is called a *LOB table space*.However, if inline LOBs are used, LOB data might be stored partly or entirely within the base table space.

An application that reads or updates a row in a table that contains LOB columns obtains its normal transaction locks on the base table. The locks on the base table also control concurrency for the LOB table space. When locks are not acquired on the base table, such as for ISO(UR), DB2 maintains data consistency by using LOB locks.

LOB locks and uncommitted read isolation

When an application uses uncommitted read isolation to read the rows, no page or row locks are taken on the base table. Therefore, these readers must take an S LOB lock to ensure that they are not reading a partial LOB or a LOB value that is inconsistent with the base row. This LOB lock is acquired and released immediately, which is sufficient for DB2 to ensure that a complete copy of the LOB data is ready for subsequent reference.

Hierarchy of LOB locks

If the LOB table space is locked with a gross lock, then LOB locks are not acquired. In a data sharing environment, the lock on the LOB table space is used to determine whether the lock on the LOB must be propagated beyond the local IRLM.

When LOB table space locks are not taken

A lock might not be acquired on a LOB table space at all. For example, if a row is deleted from a table and the value of the LOB column is null, the LOB table space that is associated with that LOB column is not locked. DB2 does not access the LOB table space when the application takes any of the following actions:

- Selects a LOB that is null or zero length
- Deletes a row where the LOB is null or zero length
- Inserts a null or zero length LOB
- Updates a null or zero-length LOB to null or zero-length

PSPI

Related concepts:

The ISOLATION (UR) option

Related tasks:

Controlling the number of LOB locks (DB2 for z/OS What's New?)

Explicitly locking LOB tables

Chapter 22, “Improving performance for LOB data,” on page 273

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

LOB and LOB table space lock modes

This information describes the modes of LOB locks and LOB table space locks.

Modes of LOB locks

PSPI

The following LOB lock modes are possible:

S (SHARE)
The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB.

X (EXCLUSIVE)
The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB.

Modes of LOB table space locks

The following lock modes are possible on the LOB table space:

IS (INTENT SHARE)
The lock owner can update LOBs to null or zero-length, or read or delete LOBs in the LOB table space. Concurrent processes can both read and change LOBs in the same table space.

IX (INTENT EXCLUSIVE)
The lock owner and concurrent processes can read and change data in the LOB table space. The lock owner acquires a LOB lock on any data it accesses.

S (SHARE)
The lock owner and any concurrent processes can read and delete LOBs in the LOB table space. An S-lock is only acquired on a LOB in the case of an ISO(UR)

SIX (SHARE with INTENT EXCLUSIVE)
The lock owner can read and change data in the LOB table space. If the lock owner is inserting (INSERT or UPDATE), the lock owner obtains a LOB lock. Concurrent processes can read or delete data in the LOB table space (or update to a null or zero-length LOB).

X (EXCLUSIVE)
The lock owner can read or change LOBs in the LOB table space. The lock owner does not need LOB locks. Concurrent processes cannot access the data.

PSPI

LOB lock and LOB table space lock duration

This information describes the duration of LOB locks and LOB table space locks.

The duration of LOB locks

PSPI

Locks on LOBs are taken when they are needed for an INSERT or UPDATE operations and released immediately at the completion of the operation. LOB locks are not held for SELECT and DELETE operations. In the case of an application that uses the uncommitted read option, a LOB lock might be acquired, but only to test the LOB for completeness. The lock is released immediately after it is acquired.

The duration of LOB table space locks

Locks on LOB table spaces are acquired when they are needed. When the table space lock is released is determined by a combination of factors:

- The RELEASE option of bind.
- Whether the SQL statement is static or dynamic.
- Whether there are held cursors or held locators.

When the RELEASE(COMMIT) option is used, the lock is released at the next commit point, unless there are held cursors or held locators. If the RELEASE(DEALLOCATE) option is used, the lock is released when the object is deallocated (the application ends). The RELEASE bind option has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you use dynamic statement caching.

PSPI

Related concepts:

The duration of a lock

Related tasks:

Choosing a RELEASE option

Related reference:

RELEASE bind option (DB2 Commands)

Locks for XML data

DB2 stores XML column values in a separate XML table space. An application that reads or updates a row in a table that contains XML columns might use lock avoidance or obtain transaction locks on the base table.

Introductory concepts

XML and DB2 (Introduction to DB2 for z/OS)

XML data type (Introduction to DB2 for z/OS)

XML table spaces (Introduction to DB2 for z/OS)

PSPI

If an XML column is updated or read, the application might also acquire transaction locks on the XML table space and XML values that are stored in the XML table space. A lock that is taken on an XML value in an XML table space is called an XML lock.

In data sharing, page P-locks are acquired during insert, update, and delete operations.

In summary, the main purpose of XML locks is for managing the space used by XML data and to ensure that XML readers do not read partially updated XML data. DB2 supports multiple versions of an XML document in XML columns. The existence of multiple versions of an XML document can lead to improved concurrency through lock avoidance.

The following table shows the relationship between an operation that is performed on XML data and the associated XML table space and XML locks that are acquired. It shows the locks that are acquired for non-versioned XML data.

Table 48. Locks that are acquired for operations on XML data. This table does not account for gross locks that can be taken because of the LOCKSIZE TABLESPACE option, the LOCK TABLE statement, or lock escalation.

Operation on XML value	XML table space lock	XML lock	Comment
Read (including UR)	IS	S	Prevents storage from being reused while the XML data is being read. If XML versions are used, the S XML lock is acquired only for UR readers.
Insert	IX	X	Prevents other processes from seeing partial XML data
Delete	IX	X	To hold space in case the delete is rolled back. Storage is not reusable until the delete is committed and no other readers of the XML data exist.
Update	IS->IX	X	Operation is a delete followed by an insert.

XML locks and uncommitted read (UR) or cursor stability (CS) isolation

When an application reads rows using uncommitted read or lock avoidance, no page or row locks are taken on the base table. Therefore, these readers must take an S XML lock to ensure that they are not reading a partial XML value or an XML value that is inconsistent with the base row. When an XML lock cannot be acquired for an SQL statement with UR isolation, DB2 might need to wait for the lock. If the lock is not granted, DB2 might return SQL return code -911 or -913.

Hierarchy of XML locks

Just as page locks (or row locks) and table space locks have a hierarchical relationship, XML locks and locks on XML table spaces have a hierarchical relationship. If the XML table space is locked with a gross lock, then XML locks are not acquired. In a data sharing environment, the lock on the XML table space is used to determine whether the lock on the XML must be propagated beyond the local IRLM.

When XML table space locks are not taken

A lock might not be acquired on an XML table space at all. DB2 does not access the XML table space if the application takes any of the following actions:

- Selects an XML value that is null
- Selects from an XML table space when XML versions are used
- Deletes a row where the XML value is null
- Inserts a null XML value
- Updates an XML value to null

PSPI

Related concepts:

XML versions (DB2 Programming for XML)

The ISOLATION (CS) option

The ISOLATION (UR) option

Related tasks:

Controlling the number of XML locks

Explicitly locking XML data

Related information:

-911 (DB2 Codes)

-913 (DB2 Codes)

XML and XML table space lock modes

This information describes the modes of XML locks and XML table space locks

PSPI

S (SHARE)
The lock owner and any concurrent processes can read the locked XML data. Concurrent processes can acquire an S lock on the XML data. The purpose of the S lock is to reserve the space used by the XML data.

X (EXCLUSIVE)
The lock owner can read, update, or delete the locked XML data. Concurrent processes cannot access the XML data.

PSPI

XML lock and XML table space lock duration

This information describes the duration of XML locks and XML table space locks.

PSPI

The duration of XML locks

X-locks on XML data that are acquired for insert, update, and delete statements are usually released at commit. The duration of XML locks acquired for select statements varies depending upon isolation level, the setting of the CURRENTDATA parameter, and whether work files or multi-row fetch are used.

XML locks acquired for fetch are not normally held until commit and are either released at next fetch or at close cursor. Because XML locks for updating (INSERT, UPDATE, and DELETE) are held until commit and because locks are put on each XML column in both a source table and a target table, it is possible that a statement such as an INSERT with a fullselect that involves XML columns can accumulate many more locks than a similar statement that does not involve XML data. To prevent system problems caused by too many locks, you can:

- Ensure that you have lock escalation enabled for the XML table spaces that are involved in the INSERT. In other words, make sure that LOCKMAX is non-zero for those XML table spaces.
- Alter the XML table space to change the LOCKSIZE to TABLESPACE before executing the INSERT with fullselect.
- Increase the LOCKMAX value on the table spaces involved and ensure that the user lock limit is sufficient.
- Use LOCK TABLE statements to lock the XML table spaces.

The duration of XML table space locks

Locks on XML table spaces are acquired when they are needed. The table space lock is released according to the value specified on the RELEASE bind option (except when a cursor is defined WITH HOLD).

PSPI

Related concepts:

The duration of a lock

XML data and query performance

Related tasks:

Choosing a RELEASE option

Related reference:

RELEASE bind option (DB2 Commands)

Chapter 17. Claims and drains

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to some objects independently of any transaction locks that are held on the object.

Claims and drains apply to the following types of objects:

- Table spaces that are not partitioned.
- Data partitions.
- Indexes that are not partitioned.
- Index partitions.

Claims

PSPI Certain utilities, commands, and some ALTER, CREATE, and DROP statements can use a *claim* to take over access to objects, independently of any transaction locks that are held on the object. When an application first accesses an object, within a unit of work, it makes a claim on the object. It releases the claim at the next commit point.

Unlike a transaction lock, a claim normally does not persist past the commit point. To access the object in the next unit of work, the application must make a new claim.

A claim indicates activity on or interest in a particular page set or partition to DB2. Claims prevent drains from occurring until the claim is released.

The following table shows the three classes of claims and the actions that they allow.

Table 49. Three classes of claims and the actions that they allow

Claim class	Actions allowed
Write	Reading, updating, inserting, and deleting
Repeatable read	Reading only, with repeatable read (RR) isolation
Cursor stability read	Reading only, with read stability (RS), cursor stability (CS), or uncommitted read (UR) isolation

DB2 generates a trace record for each time period that a task holds an uncommitted read claim. The value of the LRDRTHLD subsystem parameter controls the length of the time period. DB2 also issues warning message DSNB260I when the trace record is written. PSPI

Drains

A *drain* is the action of taking control of access to an object by preventing new claims and by waiting for existing claims to be released.

A utility can drain a partition when applications are accessing it. Drain quiesce applications by allowing each to reach a commit point, but preventing them, or other applications, from making new claims. When no more claims exist, the process that drains (the drainer) controls access to the drained object. The applications that were drained can still hold transaction locks on the drained object, but they cannot make new claims until the drainer has finished.

A drainer does not always need complete control. It can drain the following combinations of claim classes:

- Only the write claim class
- Only the repeatable read claim class
- All claim classes

For example, the CHECK INDEX utility needs to drain only writers from an index space and its associated table space. RECOVER, however, must drain all claim classes from its table space. The REORG utility can drain either writers (with DRAIN WRITERS) or all claim classes (with DRAIN ALL). PSPI

How DB2 uses drain locks

PSPI

A *drain lock* prevents conflicting processes from claiming or draining the same object at the same time. Processes that drain only writers can run concurrently. However, a process that drains all claim classes cannot drain an object concurrently with any other process. To drain an object, a drainer first acquires one or more drain locks on the object, one for each claim class that it needs to drain. When the locks are in place, the drainer can begin after all processes with claims on the object have released their claims.

A drain lock also prevents new claimers from accessing an object while a drainer has control of it.

The types of drain locks correspond to the following claim classes:

- Write
- Repeatable read
- Cursor stability read

In general, after an initial claim has been made on an object by a user, no other user in the system needs a drain lock. When the drain lock is granted, no drains on the object are in process for the claim class needed, and the claimer can proceed.

The claimer of an object requests a drain lock in the following exception cases:

- A drain on the object is in process for the claim class needed. In this case, the claimer waits for the drain lock.
- The claim is the first claim on an object before its data set has been physically opened. Here, acquiring the drain lock ensures that no exception states prohibit allocating the data set.

When the claimer gets the drain lock, it makes its claim and releases the lock before beginning its processing.

PSPI

Related tasks:

Choosing an ISOLATION option

Related reference:

LONG-RUNNING READER field (LRDRTHLD subsystem parameter) (DB2 Installation and Migration)

CHECK INDEX (DB2 Utilities)

RECOVER (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

DECLARE CURSOR (DB2 SQL)

Related information:

DSNB260I (DB2 Messages)

Concurrency during REORG

You can specify certain options that might prevent timeouts and deadlocks when you run the REORG utility.

Procedure

PSPI

To improve concurrency for REORG operations:

- If you encounter timeouts or deadlocks when you use REORG with the SHRLEVEL CHANGE option, run the REORG utility with the DRAIN ALL option. The default is DRAIN WRITERS, which is done in the log phase. The specification of DRAIN ALL indicates that both writers and readers are drained when the MAXRO threshold is reached.
- If application have long running readers that do not commit often or have overlapping claims, specify the FORCE option of the REORG utility. You can specify FORCE READERS to cancel read claims or FORCE ALL to cancel both read and write claims when REORG requests a drain on the last RETRY process. If you specify FORCE NONE, no read or write claims are canceled when REORG requests a drain.
- Consider the DRAIN ALL option in environments where a lot of update activity occurs during the log phase. With this specification, no subsequent drain is required in the switch phase.

PSPI

Related concepts:

Access with REORG TABLESPACE SHRLEVEL (DB2 Utilities)

Access with REORG INDEX SHRLEVEL (DB2 Utilities)

Related reference:

REORG TABLESPACE (DB2 Utilities)

Syntax and options of the REORG TABLESPACE control statement (DB2 Utilities)

Concurrency and compatibility for REORG TABLESPACE (DB2 Utilities)

Utility operations with nonpartitioned indexes

In a nonpartitioned index, either a partitioning index or a secondary index, an entry can refer to any partition in the underlying table space.

PSPI

DB2 can process a set of entries of a nonpartitioned index that all refer to a single partition and achieve the same results as for a partition of a partitioned index. (Such a set of entries is called a *logical partition* of the nonpartitioned index.)

Suppose that two LOAD jobs execute concurrently on different partitions of the same table space. When the jobs proceed to build a partitioned index, either a partitioning index or a secondary index, they operate on different partitions of the index and can operate concurrently. Concurrent operations on different partitions are possible because the index entries in an index partition refer only to data in the corresponding data partition for the table.

Utility processing can be more efficient with partitioned indexes because, with the correspondence of index partitions to data partitions, they promote partition-level independence. For example, the REORG utility with the PART option can run faster and with more concurrency when the indexes are partitioned. REORG rebuilds the parts for each partitioned index during the BUILD phase, which can increase parallel processing and reduce the lock contention of nonpartitioned indexes.

Similarly, for the LOAD PART and REBUILD INDEX PART utilities, the parts for each partitioned index can be built in parallel during the BUILD phase, which reduces lock contention and improves concurrency. The LOAD PART utility also processes partitioned indexes with append logic, instead of the insert logic that it uses to process nonpartitioned indexes, which also improves performance.

PSPI

Related concepts:

Rebuilding index partitions (DB2 Utilities)

Related tasks:

Loading partitions (DB2 Utilities)

Related reference:

LOAD (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

REORG INDEX (DB2 Utilities)

Utility locks on the catalog and directory

When the target of a utility is an object in the catalog or directory, such as a catalog table, the utility either drains or claims the object.

PSPI

When the target is a user-defined object, the utility claims or drains it but also uses the directory and, perhaps, the catalog; for example, to check authorization. In those cases, the utility uses transaction locks on catalog and directory tables. It acquires those locks in the same way as an SQL transaction does. For information about the SQL statements that require locks on the catalog, see Objects that are subject to locks.

PSPI

Concurrency and compatibility of utilities

Two utilities are considered *compatible* if they do not need access to the same object at the same time in incompatible modes. The concurrent operation of two utilities is not typically controlled by either drain locks or transaction locks, but merely by a set of compatibility rules.

PSPI

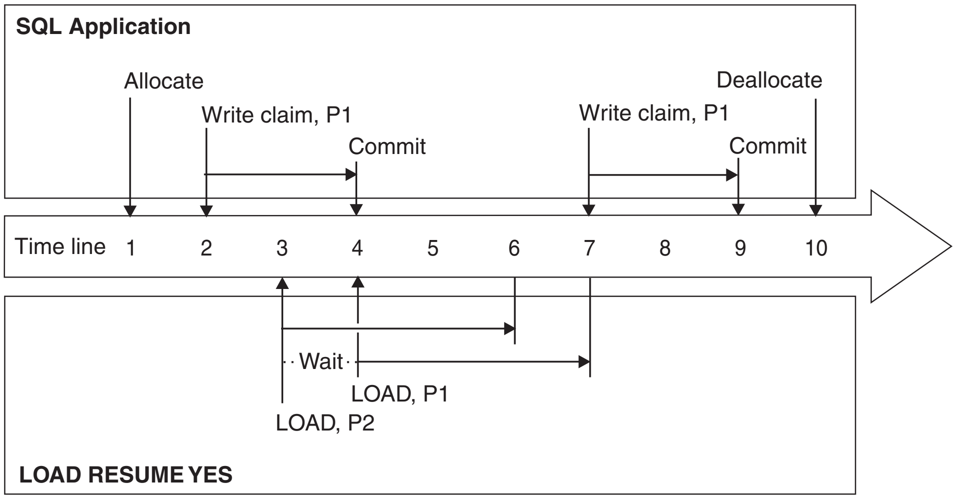
Before a utility starts, it is checked against all other utilities running on the same target object. The utility starts only if all the others are compatible.

The check for compatibility obeys the following rules:

- The check is made for each target object, but only for target objects. Typical utilities access one or more table spaces or indexes, but if two utility jobs use none of the same target objects, the jobs are always compatible.

  An exception is a case in which one utility must update a catalog or directory table space that is not the direct target of the utility. For example, the LOAD utility on a user table space updates DSNDB06.SYSCOPY. Therefore, other utilities that have DSNDB06.SYSCOPY as a target might not be compatible.
- Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions in the same table or index space are compatible.
- When two utilities access the same target object, their most restrictive access modes determine whether they are compatible. For example, if utility job 1 reads a table space during one phase and writes during the next, it is considered a writer. It cannot start concurrently with utility 2, which allows only readers on the table space. (Without this restriction, utility 1 might start and run concurrently with utility 2 for one phase; but then it would fail in the second phase, because it could not become a writer concurrently with utility 2.)

The following figure illustrates how SQL applications and DB2 utilities can operate concurrently on separate partitions of the same table space.



Time	Event
t1	An SQL application obtains a transaction lock on every partition in the table space. The duration of the locks extends until the table space is deallocated.
t2	The SQL application makes a write claim on data partition 1 and index partition 1.
t3	The LOAD jobs begin draining all claim classes on data partitions 1 and 2 and index partitions 1 and 2. LOAD on partition 2 operates concurrently with the SQL application on partition 1. LOAD on partition 1 waits.
t4	The SQL application commits, releasing its write claims on partition 1. LOAD on partition 1 can begin.
t6	LOAD on partition 2 completes.
t7	LOAD on partition 1 completes, releasing its drain locks. The SQL application (if it has not timed out) makes another write claim on data partition 1.
t10	The SQL application deallocates the table space and releases its transaction locks.

Figure 13. SQL and utility concurrency. Two LOAD jobs execute concurrently on two partitions of a table space

PSPI

Related concepts:

- Utility concurrency and compatibility (DB2 Utilities)
- DB2 online utilities (DB2 Utilities)

Related reference:

- Concurrency and compatibility for COPY (DB2 Utilities)
- Concurrency and compatibility for LOAD (DB2 Utilities)
- Concurrency and compatibility for REORG TABLESPACE (DB2 Utilities)

Part 5. Designing databases for performance

You can implement certain database design elements to improve DB2 performance, especially in special situations.

Related concepts:

- Implementing your database design (DB2 Administration Guide)
- Altering your database design (DB2 Administration Guide)

Related information:

- Designing a database (DB2 Administration Guide)

Chapter 18. Choosing data page sizes

DB2 provides many options for data page sizes.

About this task

The size of the data page is determined by the buffer pool in which you define the table space. For example, a table space that is defined in a 4 KB buffer pool has 4 KB page sizes, and one that is defined in an 8 KB buffer pool has 8 KB page sizes.

Data in table spaces is stored and allocated in record segments. Any record segment can be 4 KB in size, or the size determined by the buffer pool (4 KB, 8 KB, 16 KB, or 32 KB). In a table space with 4 KB record segments, an 8 KB page size requires two 4 KB records, and a 32 KB page size requires eight 4 KB records.

Procedure

To choose data page sizes, use the following approaches:

- Use the default of 4 KB page sizes as a starting point when access to the data is random and only a few rows per page are needed. If row sizes are very small, using the 4 KB page size is recommended.
- Use larger page sizes in the following situations:

  When the size of individual rows is greater than 4 KB

  In this case, you must use a larger page size. When considering the size of work file table spaces, remember that some SQL operations, such as joins, can create a result row that does not fit in a 4 KB page. Therefore, having at least one work file that has 32 KB pages is recommended. (Work files cannot use 8 KB or 16 KB pages.)

  When you can achieve higher density on disk by choosing a larger page size

  For example, only one 2100-byte record can be stored in a 4 KB page, which wastes almost half of the space. However, storing the record in a 32 KB page can significantly reduce this waste. The downside with this approach is the potential of incurring higher buffer pool storage costs or higher I/O costs—if you only affect a few rows, you are bringing a bigger chunk of data from disk into the buffer pool.

  Using 8 KB or 16 KB page sizes can let you store more data on your disk with less impact on I/O and buffer pool storage costs. If you use a larger page size and access is random, you might need to go back and increase the size of the buffer pool to achieve the same read-hit ratio you do with the smaller page size.

  When a larger page size can reduce data sharing overhead

  One way to reduce the cost of data sharing is to reduce the number of times the coupling facility must be accessed. Particularly for sequential processing, larger page sizes can reduce this number. More data can be returned on each access of the coupling facility, and fewer locks must be taken on the larger page size, further reducing coupling facility interactions.

If data is returned from the coupling facility, each access that returns more data is more costly than those that return smaller amounts of data, but, because the total number of accesses is reduced, coupling facility overhead is reduced.

For random processing, using an 8 KB or 16 KB page size instead of a 32 KB page size might improve the read-hit ratio to the buffer pool and reduce I/O resource consumption.

Related tasks:

Assigning database objects to buffer pools

Tuning database buffer pools

Determining the page size and data set size for DSN1PRNT (DB2 Utilities)

Related information:

Implementing DB2 table spaces (DB2 Administration Guide)

Chapter 19. Designing databases for concurrency

By following general recommendations and best practices for database design you can ensure improved concurrency on your DB2 system.

Procedure

PSPI

To design your database to promote concurrency:

- Keep like things together in the database. You can use the following approaches to accomplish these goals:
  - Create tables that are relevant to the same application in the same database.
  - Provide a private database for any application that creates private tables.
  - Create tables together in a segmented table space if they are similar in size and can be recovered together.
- Keep unlike things apart from each other in the database. You can accomplish this goal by using an adequate number of databases, schema or authorization-ID qualifiers, and table spaces to keep unlike things apart. Concurrency and performance is improved for SQL data definition statements, GRANT statements, REVOKE statements, and utilities. A general guideline is a maximum of 50 tables per database. For example, assume that user A owns table A and user B owns table B. By keeping table A and table B in separate databases, you can create or drop indexes on these two tables at the same time without causing lock contention.
- Plan for batch inserts. If your application does sequential batch insertions, excessive contention on the space map pages for the table space can occur.

  This problem is especially apparent in data sharing, where contention on the space map means the added overhead of page P-lock negotiation. For these types of applications, consider using the MEMBER CLUSTER option of CREATE TABLESPACE. This option causes DB2 to disregard the clustering index (or implicit clustering index) when assigning space for the SQL INSERT statement.
- Use LOCKSIZE ANY or PAGE as a design default. Consider LOCKSIZE ROW only when applications encounter significant lock contention, including deadlock and timeout.

  LOCKSIZE ANY is the default for CREATE TABLESPACE. It allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE and LOCKMAX SYSTEM for non-LOB/non-XML table spaces. For LOB table spaces, DB2 chooses LOCKSIZE LOB and LOCKMAX SYSTEM. Similarly, for XML table spaces, DB2 chooses LOCKSIZE XML and LOCKMAX SYSTEM.

  Page-level locking generally results in fewer requests to lock and unlock data for sequential access and manipulation, which translates to reduced CPU cost. Page-level locking is also more likely to result in sequentially inserted rows in the same data page. Row-level locking with MAXROWS=1 can suffer from data page p-locks in data sharing environments. However, page-level locking can avoid the data page p-locks when MAXROWS=1.

  Row-level locking provides better concurrency because the locks are more granular. However, the cost of each lock and unlock request is roughly the same for both page and row-level locking. Therefore, row-level locking is likely to incur additional CPU cost. Row-level locking might also result in more data page latch contention. Sequentially inserted rows, by concurrent threads, are less likely to be in the same data page under row-level locking.

- Examine small tables, looking for opportunities to improve concurrency by reorganizing data or changing the locking approach. For small tables with high concurrency requirements, estimate the number of pages in the data and in the index. In this case, you can spread out your data to improve concurrency, or consider it a reason to use row locks. If the index entries are short or they have many duplicates, then the entire index can be one root page and a few leaf pages.
- Partition secondary indexes to promote partition independence and reduce lock contention. By using data-partitioned secondary indexes (DPSIs) you might also improve index availability, especially for utility processing, partition-level operations (such as dropping or rotating partitions), and recovery of indexes.

  However, using data-partitioned secondary indexes does not always improve the performance of queries. For example, for a query with a predicate that references only the columns of a data-partitioned secondary index, DB2 must probe each partition of the index for values that satisfy the predicate if index access is chosen as the access path. Therefore, take into account data access patterns and maintenance practices when deciding to use a data-partitioned secondary index. Replace a nonpartitioned index with a partitioned index only if you will realize perceivable benefits such as improved data or index availability, easier data or index maintenance, or improved performance.
- Store fewer rows of data in each data page. You can use the MAXROWS clause of CREATE or ALTER TABLESPACE, to specify the maximum number of rows that can be on a page. For example, if you use MAXROWS 1, each row occupies a whole page, and you confine a page lock to a single row. Consider this option if you have a reason to avoid using row locking, such as in a data sharing environment where row locking overhead can be greater.
- If multiple applications access the same table, consider defining the table as VOLATILE. DB2 uses index access whenever possible for volatile tables, even if index access does not appear to be the most efficient access method because of volatile statistics. Because each application generally accesses the rows in the table in the same order, lock contention can be reduced.

What to do next

For DB2 subsystems that are members of data sharing groups additional recommendations apply. For information about improving concurrency in data sharing groups, see Improving concurrency in data sharing environments (DB2 Data Sharing Planning and Administration).

PSPI

Related tasks:

Improving concurrency

Programming for concurrency

Analyzing concurrency

Specifying the maximum number of locks that a single process can hold

The LOCKS PER USER field of installation panel DSNTIPJ specifies the maximum number of page, row, LOB, or XML locks that can be held by a single process at any one time. This field includes locks for both the DB2 catalog and directory and for user data. However, it does not include catalog locks for bind operations.

About this task

PSPI

When a request for a page, row, LOB, or XML lock exceeds the specified limit, it receives SQLCODE -904: “resource unavailable” (SQLSTATE '57011'). The requested lock cannot be acquired until some of the existing locks are released.

The default value is 10 000.

The default should be adequate for 90 percent of the workload when using page locks. If you use row locks on very large tables, you might want a higher value. If you use LOBs or XML data, you might need a higher value.

Procedure

To determine the maximum number of locks that a process requires:

- Review application processes that require higher values to see if they can use table space locks rather than page, row, LOB, or XML locks. The accounting trace shows the maximum number of page, row, LOB, or XML locks a process held while an application runs.
- Remember that the value specified is for a single application. Each concurrent application can potentially hold up to the maximum number of locks specified. Do not specify zero or a very large number unless that number is required to run your applications.

PSPI

Specifying the size of locks for a table space

The LOCKSIZE clause of CREATE and ALTER TABLESPACE statements specifies the size for locks held on a table or table space by application processes.

About this task

PSPI

You can use the ALTER TABLESPACE statement to change the lock size for user data. You can also change the lock size of any DB2 catalog table space that is not a LOB table space.The relevant options are:

LOCKSIZE TABLESPACE
A process acquires no table, page, row, LOB, or XML locks within the table space. That improves performance by reducing the number of locks maintained, but greatly inhibits concurrency.

LOCKSIZE TABLE
A process acquires table locks on tables in a segmented table space without partitions. If the table space contains more than one table, this option can provide acceptable concurrency with little extra cost in processor resources.

LOCKSIZE PAGE
A process acquires page locks, plus table, partition, or table space locks of modes that permit page locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table, partition, or table space lock of mode S or X, without page locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under "Lock promotion" on page 210.

LOCKSIZE ROW
A process acquires row locks, plus table, partition, or table space locks of modes that permit row locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table, partition, or table space lock of mode S or X, without row locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under "Lock promotion" on page 210.

LOCKSIZE ANY
DB2 chooses the size of the lock, usually LOCKSIZE PAGE. For LOB table spaces DB2 usually chooses LOCKSIZE LOB.

LOCKSIZE LOB
If a LOB must be accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX). This option is valid only for LOB table spaces. See Locks for LOB data for more information about LOB locking.

LOCKSIZE XML
If XML must be accessed, a process acquires XML locks and the necessary XML table space locks (IS or IX). This option is valid only for XML table spaces. See Locks for XML data for more information about XML locking.

DB2 attempts to acquire an S lock on table spaces that are started with read-only access. If the LOCKSIZE is PAGE, ROW, or ANY and DB2 cannot get the S lock, it requests an IS lock. If a partition is started with read-only access, DB2 attempts to get an S lock on the partition that is started RO. For a complete description of how the LOCKSIZE clause affects lock attributes, see "How DB2 chooses lock types" on page 205.

The default option is LOCKSIZE ANY, and the LOCKRULE column of the SYSIBM.SYSTABLESPACE catalog table records the current value for each table space.

If you do not use the default, base your choice upon the results of monitoring applications that use the table space.

Procedure

The question of whether to use row or page locks depends on your data and your applications, and no single choice is best for every case. Consider the following trade-offs when deciding which LOCKSIZE option to use for a table space:

- Generally, use LOCKSIZE PAGE as a design default. Page-level locking generally results in better CPU time because fewer lock and unlock requests are required for sequential data access and manipulation. The amount of resources required to acquire, maintain, and release each lock is about the same in both row-level and page-level locking. Therefore, when page-level locking is used, a table space or index scan of data that has 10 rows per page might require only one-tenth of the CPU resource that would be used for row-level locking. Under page-level locking, sequentially inserted rows are also more likely to be in the same data page.

- Use LOCKSIZE ROW when applications that access a table space encounter significant numbers of deadlock and timeout situations under LOCKSIZE PAGE. Locking single rows instead of entire pages, might reduce the chances of contention with other process by 90%, especially when access is random. (Row locking is not recommended for sequential processing.)

  Lock avoidance is important when row-level locking is used. Therefore, the recommendation is for applications to use ISOLATION(CS) and CURRENTDATA(NO) bind options whenever possible. In many cases, DB2 can avoid acquiring a lock when reading data that is known to be committed. Consequently, if only 2 of 10 rows on a page contain uncommitted data, DB2 must lock the entire page when using page lock. However, DB2 but might ask for locks on only the two uncommitted rows when using row-level locks. In that case, the resource required for row locks would be only twice as much, not 10 times as much, as that required for page-level locks.

  However, if two applications update the same rows of a page, and not in the same sequence, then row locking might even increase contention. With page locks, the second application to access the page must wait for the first to finish and might time out. With row locks, the two applications can access the same page simultaneously. However, they might encounter a deadlock while trying to access the same set of rows.
- When considering changing the lock size for a DB2 catalog table space, remember that internal processes such as bind, authorization checking, and utility processing might also access the catalog.

PSPI

Related concepts:

Lock size

Lock escalation

Related tasks:

Choosing an ISOLATION option

Choosing a CURRENTDATA option

Related reference:

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

CURRENTDATA bind option (DB2 Commands)

ISOLATION bind option (DB2 Commands)

Specifying the maximum number of locks that a process can hold on a table space

You can specify the LOCKMAX clause of the CREATE and ALTER TABLESPACE statements for tables of user data and also for tables in the DB2 catalog, by using ALTER TABLESPACE.

About this task

PSPI

The values of the LOCKMAX clause have the following meanings:

LOCKMAX *n*

Specifies the maximum number of page or row locks that a single application process can hold on the table space before those locks are escalated. For LOB table spaces, this value specifies the number of LOB locks that the application process can hold before escalating. For XML table spaces, this value specifies the number of XML locks that the application process can hold before escalating. For an application that uses Sysplex query parallelism, a lock count is maintained on each member.

LOCKMAX SYSTEM

Specifies that *n* is effectively equal to the system default set by the field LOCKS PER TABLE(SPACE) of installation panel DSNTIPJ.

LOCKMAX 0

Disables lock escalation entirely.

The default value depends on the value of LOCKSIZE, as shown in the following table.

Table 50. How the default for LOCKMAX is determined

LOCKSIZE	Default for LOCKMAX
ANY	NUMLKTS subsystem parameter value.
other	0

Note: For XML table spaces, the default value of LOCKMAX is inherited from the base table space.

Catalog record: Column LOCKMAX of table SYSIBM.SYSTABLESPACE.

Procedure

Use one of the following approaches if you do not use the default value:

- Base your choice upon the results of monitoring applications that use the table space.
- Aim to set the value of LOCKMAX high enough that, when lock escalation occurs, one application already holds so many locks that it significantly interferes with others. For example, if an application holds half a million locks on a table with a million rows, it probably already locks out most other applications. Yet lock escalation can prevent it from potentially acquiring another half million locks.
- If you alter a table space from LOCKSIZE PAGE or LOCKSIZE ANY to LOCKSIZE ROW, consider increasing LOCKMAX to allow for the increased number of locks that applications might require.

PSPI

Related concepts:

Lock size

Related tasks:

Specifying a default value for the LOCKMAX option

Related reference:

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

LOCKS PER TABLE(SPACE) field (NUMLKTS subsystem parameter) (DB2 Installation and Migration)

SYSIBM.SYSTABLESPACE table (DB2 SQL)

Related information:

-904 (DB2 Codes)

Controlling the number of LOB locks

You can control the number of LOB locks that are taken.

About this task

Introductory concepts

Large object data types (Introduction to DB2 for z/OS)
Large object table spaces (Introduction to DB2 for z/OS)

PSPI

LOB locks are counted toward the total number of locks allowed per user.

Procedure

To control the number of LOB locks, use any of the following approaches:

- Set the value of the NUMLKUS subsystem parameter. The number of LOB locks that are acquired during a unit of work is reported in IFCID 0020.
- Use the LOCKMAX clause of the CREATE TABLESPACE or ALTER TABLESPACE statement to control the number of LOB locks that are acquired within a particular LOB table space.

PSPI

Related concepts:

Locks for LOB data

Large objects (LOBs) (DB2 SQL)

Related reference:

LOCKS PER USER field (NUMLKUS subsystem parameter) (DB2 Installation and Migration)

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

Controlling lock size for LOB table spaces

You can use the LOCKSIZE option control the size of locks that are acquired when applications access data in LOB table spaces.

About this task

PSPI

The LOCKSIZE TABLE, PAGE, ROW, and XML options are not valid for LOB table spaces. The other options act as follows:

LOCKSIZE TABLESPACE
A process acquires no LOB locks

LOCKSIZE ANY
DB2 chooses the size of the lock. For a LOB table space, this is usually LOCKSIZE LOB.

LOCKSIZE LOB
If LOBs are accessed, a process acquires the necessary LOB table space locks (IS or IX), and might acquire LOB locks.

PSPI

Controlling the number of XML locks

You can control the number of XML locks that are taken.

About this task

PSPI

XML locks are counted toward the total number of locks allowed per user.

Procedure

To control the number of XML locks, use the following approaches:

- Specify the value of the NUMLKUS subsystem parameter. The number of XML locks that are acquired during a unit of work is reported in IFCID 0020.

  PSPI

- Use the LOCKMAX clause of the CREATE TABLESPACE or ALTER TABLESPACE statement to control the number of LOB locks that are acquired within a particular XML table space.

Related concepts:

Locks for XML data

XML data type (Introduction to DB2 for z/OS)

XML table spaces (Introduction to DB2 for z/OS)

Related reference:

LOCKS PER USER field (NUMLKUS subsystem parameter) (DB2 Installation and Migration)

ALTER TABLESPACE (DB2 SQL)

CREATE TABLESPACE (DB2 SQL)

Specifying the size of locks for XML data

You can specify that DB2 uses table space or XML locks for XML data.

About this task

Introductory concepts

XML table spaces (Introduction to DB2 for z/OS)

Procedure

To control the size of locks for XML data:

Specify the LOCKSIZE clause when you alter the table space:

`LOCKSIZE TABLESPACE`
A process acquires no XML locks.

`LOCKSIZE XML`
If XML data is accessed, a process acquires XML locks and the necessary XML table space locks (IS or IX).

PSPI

Related concepts:

Lock size

Related tasks:

Specifying the size of locks for a table space

Related reference:

ALTER TABLESPACE (DB2 SQL)

Controlling XML lock escalation

You can use the LOCKMAX clause of the ALTER TABLESPACE statement to control the number of locks that are acquired within a particular XML table space before the lock is escalated.

Procedure

PSPI

Specify the LOCKMAX option of the ALTER TABLESPACE statement. When the total number of page, row, and XML locks reaches the maximum that you specify in the LOCKMAX clause, the XML locks escalate to a gross lock on the XML table space, and the XML locks are released
Information about XML locks and lock escalation is reported in IFCID 0020.

PSPI

Related concepts:

Locks for XML data

XML data type (Introduction to DB2 for z/OS)

XML table spaces (Introduction to DB2 for z/OS)

Related reference:

ALTER TABLESPACE (DB2 SQL)

Specifying a default value for the LOCKMAX option

The NUMLKTS subsystem parameter specifies the default value (at the subsystem level) for the LOCKMAX clause of CREATE TABLESPACE and ALTER TABLESPACE statements.

Procedure

- Use the default or, if you are migrating from a previous release of DB2, continue to use the existing value. The value should be less than the value for the NUMLKUS subsystem parameter, unless the value of the NUMLKUS subsystem parameter is 0.
- When you create or alter a table space, especially when you alter one to use row locks, use the LOCKMAX clause explicitly for that table space.

PSPI

Related tasks:

Specifying the maximum number of locks that a process can hold on a table space

Related reference:

LOCKS PER TABLE(SPACE) field (NUMLKTS subsystem parameter) (DB2 Installation and Migration)

LOCKS PER USER field (NUMLKUS subsystem parameter) (DB2 Installation and Migration)

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

Improving concurrency for update and delete operations

You can avoid certain deadlock situations by controlling the lock modes that are used by certain SELECT, UPDATE, and DELETE statements.

About this task

Statements that search for data to update or delete can encounter lock contention, such as timeout and deadlock, when concurrent operations acquire locks on the data during the search phase. The contention happens when locks from concurrent processes prevent the statement from acquiring the locks that are required for the update or delete operations.

For example, if a statement uses S-locks while it searches for data to update, a concurrent operation can acquire S locks on the same data. The S-lock from the concurrent operation might prevent the first statement from acquiring the X-lock that it must use to update or delete the data. The result might be a timeout. If both concurrent operations need to acquire x-locks, deadlock situations can result.

Procedure

To improve concurrency for statements that search before they update or delete data, use the following approaches:

- Specify USE AND KEEP *lock-mode* LOCKS in the isolation-clause. This clause applies at the statement level to SELECT statements that use RR or RS isolation. The best mode to specify depends on the filter factor of the search operation:

  - Specify U-locks when the filter factor is high and a large percentage of the rows that are fetched for the statement are updated or deleted.
  - Specify X-locks if almost all fetched rows are updated or deleted.
- Set the value of the RRULOCK subsystem parameter. This value controls the mode of locks at the subsystem level for cursor-based SELECT statements that specify FOR UPDATE. It also applies to UPDATE and DELETE statements when a cursor is not used. The YES option can avoid deadlocks but it reduces concurrency. This option applies only to statements in packages that use RR or RS isolation.

Results

When USE AND KEEP *lock-mode* LOCKS is specified in a statement, it overrides the value of the RRULOCK subsystem parameter. Another subsystem parameter XLKUPDLT also controls lock modes for non-cursor UPDATE and DELETE statements. The value of the XLKUPDLT subsystem parameter overrides the value of the RRULOCK subsystem parameter. It can be used to reduce the cost of lock requests for UPDATE and DELETE operations in data sharing environments.

The following table summarizes the results of the options.

Table 51. Lock modes for statements that use RR and RS options

Option	**cursor SELECT with FOR UPDATE**	**SELECT with RS or RR isolation**	**UPDATE or DELETE**
USE AND KEEP *lock-mode* LOCKS	S, U, or X-locks, as specified by *lock-mode*.	S, U, or X-locks, as specified by *lock-mode*.	Not applicable
RRULOCK=YES	U-locks	Not applicable.	U-locks
XLKUPDLT=YES	Not applicable	Not applicable	X-locks

Related concepts:

Lock modes

The ISOLATION (RR) option

The ISOLATION (RS) option

Related tasks:

Disabling update locks for searched UPDATE and DELETE (DB2 Data Sharing Planning and Administration)

Related reference:

U LOCK FOR RR/RS field (RRULOCK subsystem parameter) (DB2 Installation and Migration)

X LOCK FOR SEARCHED U/D field (XLKUPDLT subsystem parameter) (DB2 Installation and Migration)

isolation-clause (DB2 SQL)

Avoiding locks during predicate evaluation

The EVALUATE UNCOMMITTED field of installation panel DSNTIP8 indicates if predicate evaluation can occur on uncommitted data of other transactions.

About this task

PSPI

The option applies only to stage 1 predicate processing that uses table access (table space scan, index-to-data access, and RID list processing) for queries with isolation level RS or CS.

Although this option influences whether predicate evaluation can occur on uncommitted data, it does not influence whether uncommitted data is returned to an application. Queries with isolation level RS or CS return only committed data. They never return the uncommitted data of other transactions, even if predicate evaluation occurs on such. If data satisfies the predicate during evaluation, the data is locked as needed, and the predicate is evaluated again as needed before the data is returned to the application.

A value of NO specifies that predicate evaluation occurs only on committed data (or on the uncommitted changes made by the application). NO ensures that all qualifying data is always included in the answer set.

A value of YES specifies that predicate evaluation can occur on uncommitted data of other transactions. With YES, data might be excluded from the answer set. Data that does not satisfy the predicate during evaluation but then, because of undo processing (ROLLBACK or statement failure), reverts to a state that does satisfy the predicate is missing from the answer set. A value of YES enables DB2 to take fewer locks during query processing. The number of locks avoided depends on the following factors:

- The query's access path
- The number of evaluated rows that do not satisfy the predicate
- The number of those rows that are on overflow pages

The default value for this field is NO.

Procedure

Specify YES to improve concurrency if your applications can tolerate returned data to falsely exclude any data that would be included as the result of undo processing (ROLLBACK or statement failure).

PSPI

Chapter 20. Organizing tables by hash for fast access to individual rows

You can organize tables for hash access to improve the performance of queries that use unique equal predicates to access individual rows.

About this task

When you create a table or alter an existing table, you can choose to organize the table by hash. You can also alter the size of the hash space on tables that are already organized by hash.

Tables that are organized by hash are efficient for access by queries that use equal predicates that are unique to individual rows in the table. When DB2 selects a hash access path, as few as one I/O might be required to retrieve the row from the table, which reduces CPU usage. However, hash access requires additional disk space to store data.

Tables that are good candidates for hash organization have the following attributes:

- DB2 uses unique value lookup in an index to access rows in the table, or the table is in memory. When you organize a table for hash access, DB2 automatically enforces uniqueness on a column or set of columns that you specify. So, you do not need to enforce uniqueness separately.
- DB2 does not use range scans to access rows in the table.
- DB2 uses random lookups to access data in the table.
- The table is most often used for queries that are satisfied by a row in the table, more often than not.
- The size of the data in the table is relatively static, or the maximum size of the data is known.
- The size of individual rows in the table do not vary greatly.
- At least 20 rows fit on a data page.

Tables that have a stable or predictable size are good candidates for hash organization. Tables that are smaller than the specified hash space do not use their allocated disk space efficiently. DB2 automatically creates an overflow index when you organize a table by hash. If a table exceeds the specified hash space, DB2 places extra rows in the overflow index. Rows that are placed in this index are not enabled for hash access, and DB2 uses the index to retrieve them.

The following restrictions apply to tables that are organized by hash:

- The following types of tables and table spaces cannot be organized for hash access:
  - LOB table spaces
  - XML table spaces
  - MQT table spaces
  - Tables that have clone tables
- Clustered indexes are not compatible with tables that are organized for hash access.

- Hash access is not used to access any table in a query block that contains a star join (JOIN_TYPE = S)
- Hash access is not used for sensitive dynamic scrollable cursor with multi-element in-list.
- Parallelism is not used for parallel groups when hash access is used.

Scrollable cursors are available on the result table of a query with an IN list predicate.

You can organize tables by hash only if the tables are in partition-by-growth or range-partitioned universal tables spaces and use reordered row format. Tables in basic row format cannot to take advantage of hash organization.

Related concepts:

Hash access (ACCESSTYPE='H', 'HN', or 'MH')

Related tasks:

Creating tables that use hash organization (DB2 Administration Guide)

Altering tables to enable hash access (DB2 Administration Guide)

Altering the size of your hash spaces (DB2 Administration Guide)

Monitoring hash access

Related information:

Hash access (DB2 10 for z/OS)

Hash access (DB2 10 for z/OS Performance Topics)

Choosing hash table candidates (DB2 10 for z/OS Performance Topics)

Managing space and page size for hash-organized tables

By managing the table space size of your hash-organized tables you can reduce the cost of accessing the index for rows that overflow the fixed hash space.

About this task

GUPI

When a table is organized for hash access, DB2 uses a hash calculation to determine which data page to place each row of data into. It also uses a separate calculation to determine where in the page to place the row. That process is somewhat, but not completely, random. Statistical variation means that sometimes too many rows are placed on the same page by the hash calculation. Rows that do not fit on the page are stored outside of the hash space. Entries are added to a hash overflow index so that the rows can be located.

Procedure

To minimize the number of hash overflows create the table space for the hash-organized table explicitly, and use the following approaches:

- Define the fixed hash space to be larger than the data that it contains to minimize the number of overflows. You can specify the AUTOESTSPACE(YES) option when you run the REORG TABLESPACE utility for the hash-organized table. When you specify that option, DB2 determines a size that results in a percentage of overflows of approximately five percent, if the amount of extra space that is needed does not exceed 50% of the size of the data. When you specify AUTOESTSPACE(YES) DB2 also uses the specified PCTFREE value for the table space when it determines how much space to allocate.

- Choose a page size the enables enough rows to fit on a page. When few rows fit on a page, hash access requires more extra space to perform well. Generally, 20 rows per page yields an acceptable amount of overflows.
- Specify the following other options for table spaces that are organized by hash:
  - Specify DEFINE YES to ensure that the fixed table space is allocated successfully before the first access.
  - Specify NUMPARTS for range-partitioned table spaces. Partition-by-growth table spaces automatically calculate NUMPARTS based on the specified HASH SPACE and DSSIZE.
  - Specify a DSSIZE value large enough to fit the hash space for each partition. Partition-by-growth table spaces use DSSIZE to validate the hash space for each partition.
  - Specify -1 for the value of PRIQTY. DB2 uses the default value for primary space allocation.
  - Do not use the MAXROWS option with tables that are organized for hash access. It is not used in the fixed hash space. However, the value when specified applies normally in the hash overflow area.

  The following values cannot be specified for tables that are organized for hash access:
  - APPEND YES
  - MEMBER CLUSTER
  - FREEPAGE

What to do next

To learn more about how you might achieve fewer overflows, or reduce the amount of extra space that is needed, see “Fine-tuning hash space and page size.”

GUPI

Related information:

Monitoring the performance of hash access tables (DB2 10 for z/OS Technical Overview)

Fine-tuning hash space and page size

You can fine-tune the amount of overhead from hash overflow by adjusting two attributes of the hash-organized table space: The amount of additional space, and the number of rows per data page.

Before you begin

GUPI

Before trying to manually tune the size of your hash space, determine whether automatic space estimation yields acceptable performance:

1. Reorganize the hash-organized table space and specify the AUTOESTSPACE(YES) option.
2. Monitor the hash access to determine whether the performance is acceptable.

About this task

The process that DB2 uses to determine the placement of a row in a hash-organized table space is somewhat, but not completely, random. Consequently, statistical variation means that sometimes too many rows are placed on the same page When that happens, rows that do not fit on the page are stored outside of the hash space, and an entry is added to a hash overflow index so that the row can be located. When that happens an *overflow* occurs. Rows that do not fit on the determined page are stored outside of the hash space, and an entry is added to a hash overflow index so that the row can be located.

By allocating additional space for table space, you can reduce the number of these overflow situations. The number of rows that fit on a page also has a significant impact on the amount of overflows.

The following table shows estimated multipliers for determining the amount of additional space to allocate, according to the number of rows that fit on page, to achieve a certain estimated percentage of overflows.

Table 52. Estimated size multipliers for fixed-size hash-organized table spaces by percentage of overflows.

Rows per page	**20% overflow**	**10% overflow**	**5% overflow**	**2.5% overflow**	**1.0% overflow**	**0.5% overflow**	**0.1% overflow**	**0.01% overflow**	**0.0001% overflow**
1	2.18	4.69	9.69	19.7	49.7	99.7	499.7	4999.7	49999.7
2	1.3	2.08	3.16	4.68	7.69	11.07	25.35	81.18	257.73
3	1.1	1.6	2.2	2.95	4.23	5.49	9.82	21.83	47.7
4	1.02	1.4	1.83	2.33	3.13	3.87	6.13	11.43	20.85
5	0.97	1.29	1.64	2.02	2.61	3.12	4.61	7.75	12.71
7	0.92	1.18	1.44	1.71	2.1	2.43	3.31	4.94	7.19
10	0.88	1.1	1.3	1.5	1.77	1.99	2.54	3.47	4.63
15	0.86	1.04	1.19	1.34	1.54	1.69	2.04	2.59	3.22
20	0.85	1.01	**1.14**	1.27	1.42	1.54	1.81	2.21	2.64
30	0.84	0.98	1.09	1.19	1.31	1.39	1.59	1.86	2.14
40	0.83	0.96	1.06	1.15	1.25	1.32	1.47	1.69	1.9
50	0.83	0.95	1.04	1.12	1.21	1.27	1.41	1.58	1.76
100	0.83	0.94	1.01	1.07	1.13	1.17	1.26	1.37	1.47
150	0.83	0.93	1	1.05	1.1	1.13	1.2	1.29	1.36
200	0.83	0.93	0.99	1.04	1.08	1.11	1.17	1.24	1.31
250	0.83	0.93	0.99	1.03	1.07	1.1	1.15	1.22	1.27

Procedure

To fine tune the size of your hash space:

- Ensure that a sufficient amount of additional space is provided. For example, if 20 rows fit on a data page, and you want approximately 5% of the rows to be hash overflows, you would multiply the size of your data by 1.14 (14% overhead) to determine the size of your hash space. Note that additional space is actually used in that case because 5% of the rows overflow, meaning a total space overhead of 19%

- Ensure that an appropriate number of rows fit on each page. For example, a table that contains 1500-byte rows in 4KB pages has two rows on each page. To achieve a overflow level of 5% would require you to specify a fixed hash space size that is 3.16 times larger than the size of the data in the table. However, if you convert the table to use 32KB pages, each page would contain 16 rows, meaning that a fixed size for the hash space between 1.14 and 1.19 times larger would be sufficient to achieve 5% overflows.

GUPI

Chapter 21. Using materialized query tables to improve SQL performance

Materialized query tables can simplify query processing, greatly improve the performance of dynamic SQL queries, and are particularly effective in data warehousing applications, where you can use them to avoid costly aggregations and joins against large fact tables.

About this task

PSPI

Materialized query tables are tables that contain information that is derived and summarized from other tables. Materialized query tables pre-calculate and store the results of queries with expensive join and aggregation operations. By providing this summary information, materialized query tables can simplify query processing and greatly improve the performance of dynamic SQL queries. Materialized query tables are particularly effective in data warehousing applications.

Automatic query rewrite is the process DB2 uses to access data in a materialized query table. If you enable automatic query rewrite, DB2 determines if it can resolve a dynamic query or part of the query by using a materialized query table.

Procedure

To take advantage of eligible materialized query tables:

Rewrite the queries to use materialized query tables instead of the underlying base tables. Keep in mind that a materialized query table can yield query results that are not current if the base tables change after the materialized query table is updated.

PSPI

Configuring automatic query rewrite

You can enable DB2 to rewrite certain queries to use materialized query tables instead of base tables for faster processing.

Procedure

PSPI

To take advantage of automatic query rewrite with materialized query tables:

1. Define materialized query tables.
2. Populate materialized query tables.
3. Refresh materialized query tables periodically to maintain data currency with base tables. However, realize that refreshing materialized query tables can be an expensive process.

4. Enable automatic query rewrite, and exploit its functions by submitting read-only dynamic queries.
5. Evaluate the effectiveness of the materialized query tables. Drop under-used tables, and create new tables as necessary.

PSPI

Materialized query tables and automatic query rewrite

As the amount of data has increased, so has the demand for more interactive queries.

PSPI

Because databases have grown substantially over the years, queries must operate over huge amounts of data. For example, in a data warehouse environment, decision-support queries might need to operate over 1 to 10 terabytes of data, performing multiple joins and complex aggregation. As the amount of data has increased, so has the demand for more interactive queries.

Despite the increasing amount of data, these queries still require a response time in the order of minutes or seconds. In some cases, the only solution is to pre-compute the whole or parts of each query. You can store these pre-computed results in a materialized query table. You can then use the materialized query table to answer these complicated queries when the system receives them. Using a process called automatic query rewrite, DB2 recognizes when it can transparently rewrite a submitted query to use the stored results in a materialized query table. By querying the materialized query table instead of computing the results from the underlying base tables, DB2 can process some complicated queries much more efficiently. If the estimated cost of the rewritten query is less than the estimated cost of the original query, DB2 uses the rewritten query.

Automatic query rewrite

When it uses *Automatic query rewrite*, DB2 compares user queries with the fullselect query that defined a materialized query table. It then determines whether the contents of a materialized query table overlap with the contents of a query. When an overlap exists, the query and the materialized query table are said to *match*. After discovering a match, DB2 rewrites the query to access the matched materialized query table instead of the one or more base tables that the original query specified. If a materialized query table overlaps with only part of a query, automatic query rewrite can use the partial match. Automatic query rewrite compensates for the non-overlapping parts of the query by accessing the tables that are specified in the original query.

Automatic query rewrite tries to search for materialized query tables that result in an access path with the lowest cost after the rewrite. DB2 compares the estimated costs of the rewritten query and of the original query and chooses the query with the lower estimated cost.

Example

Suppose that you have a very large table named TRANS that contains one row for each transaction that a certain company processes. You want to tally the total amount of transactions by some time period. Although the table contains many columns, you are most interested in these four columns:

- YEAR, MONTH, DAY, which contain the date of a transaction
- AMOUNT, which contains the amount of the transaction

To total the amount of all transactions between 2001 and 2006, by year, you would use the following query:

```
SELECT YEAR, SUM(AMOUNT)
  FROM TRANS
  WHERE YEAR >= '2001' AND YEAR <= '2006'
  GROUP BY YEAR
  ORDER BY YEAR;
```

This query might be very expensive to run, particularly if the TRANS table is a very large table with millions of rows and many columns.

Now suppose that you define a materialized query table named STRANS by using the following CREATE TABLE statement:

```
CREATE TABLE STRANS AS
  (SELECT YEAR AS SYEAR,
          MONTH AS SMONTH,
          DAY AS SDAY,
          SUM(AMOUNT) AS SSUM
     FROM TRANS
     GROUP BY YEAR, MONTH, DAY)
     DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

After you populate STRANS with a REFRESH TABLE statement, the table contains one row for each day of each month and year in the TRANS table.

Using the automatic query rewrite process, DB2 can rewrite the original query into a new query. The new query uses the materialized query table STRANS instead of the original base table TRANS:

```
SELECT SYEAR, SUM(SSUM)
  FROM STRANS
  WHERE SYEAR >= '2001' AND SYEAR <= '2006'
  GROUP BY SYEAR
  ORDER BY SYEAR
```

If you maintain data currency in the materialized query table STRANS, the rewritten query provides the same results as the original query. The rewritten query offers better response time and requires less CPU time.

GUPI

Queries that are eligible for rewrite

DB2 supports automatic query rewrite only for read-only, dynamic queries that do not contain parameter markers. DB2 cannot automatically rewrite statically bound queries.

PSPI

You can always refer to a materialized query table explicitly in a statically bound query or in a dynamically prepared query. However, you should consider updating materialized query table data more frequently if you frequently query the table directly. Also, allowing users to refer directly to materialized query tables reduces an installation's flexibility of dropping and creating materialized query tables without affecting applications.

PSPI

How DB2 considers automatic query rewrite

In general, DB2 considers automatic query rewrite at the query block level. A read-only, dynamic query can contain multiple query blocks.

PSPI

For example, it might contain a subselect of UNION or UNION ALL, temporarily materialized views, materialized table expressions, and subquery predicates. DB2 processes the query without automatic query rewrite if the query block is or contains any of the following items:

- A fullselect in the UPDATE SET statement.
- A fullselect in the INSERT statement.
- A fullselect in the materialized query table definition in the REFRESH TABLE statement.
- An outer join.
- A query block that contains user-defined scalar or table functions with the EXTERNAL ACTION attribute or the NON-DETERMINISTIC attribute, or with the built-in function RAND.
- Parameter markers

If none of these items exist in the query block, DB2 considers automatic query rewrite. DB2 analyzes the query block in the user query and the fullselect in the materialized query table definition to determine if it can rewrite the query. The materialized query table must contain all of the data from the source tables (in terms of both columns and rows) that DB2 needs to satisfy the query. For DB2 to choose a rewritten query, the rewritten query must provide the same results as the user query. (DB2 assumes the materialized query table contains current data.) Furthermore, the rewritten query must offer better performance than the original user query.

DB2 performs a sophisticated analysis to determine whether it can obtain the results for the query from a materialized query table:

- DB2 compares the set of base tables that were used to populate the materialized query table to the set of base tables that are referenced by the user query. If these sets of tables share base tables in common, the query is a candidate for query rewrite.
- DB2 compares the predicates in the materialized query table fullselect to the predicates in the user query. The following factors influence the comparison:
  - The materialized query table fullselect might contain predicates that are not in the user query. If so, DB2 assumes that these predicates might have resulted in discarded rows when the materialized query table was refreshed. Thus, any rewritten query that makes use of the materialized query table might not give the correct results. The query is not a candidate for query rewrite.

    Exception: DB2 behavior differs if a predicate joins a common base table to an extra table that is unique to the materialized query table fullselect. The predicate does not result in discarded data if you define a referential constraint between the two base tables to make the predicate *lossless*. However, the materialized query table fullselect must not have any local predicates that reference this extra table.

    For an example of a lossless predicate, see Example 2 under "Automatic query rewrite—complex examples."

  - Referential constraints on the source tables are very important in determining whether automatic query rewrite uses a materialized query table.
  - Predicates are much more likely to match if you code the predicate in the user query so that it is the same or very similar to the predicate in the materialized query table fullselect. Otherwise, the matching process might fail on some complex predicates.

    For example, the matching process between the simple equal predicates such as `COL1 = COL2` and `COL2 = COL1` succeeds. Furthermore, the matching process between simple equal predicates such as `COL1 * (COL2 + COL3) = COL5` and `COL5 = (COL3 + COL2) * COL1` succeeds. However, the matching process between equal predicates such as `(COL1 + 3) * 10 = COL2` and `COL1 * 10 + 30 = COL2` fails.
  - The items in an IN-list predicate do not need to be in exactly the same order for predicate matching to succeed.
- DB2 compares GROUP BY clauses in the user query to GROUP BY clauses in the materialized query table fullselect. If the user query requests data at the same or higher grouping level as the data in the materialized query table fullselect, the materialized query table remains a candidate for query rewrite. DB2 uses functional dependency information and column equivalence in this analysis.
- DB2 compares the columns that are requested by the user query with the columns in the materialized query table. If DB2 can derive the result columns from one or more columns in the materialized query table, the materialized query table remains a candidate for query rewrite.DB2 uses functional dependency information and column equivalence in this analysis.
- DB2 examines predicates in the user query that are not identical to predicates in the materialized query table fullselect. Then, DB2 determines if it can derive references to columns in the base table from columns in the materialized query table instead. If DB2 can derive the result columns from the materialized query table, the materialized query table remains a candidate for query rewrite.

If all of the preceding analyses succeed, DB2 rewrites the user query. DB2 replaces all or some of the references to base tables with references to the materialized query table. If DB2 finds several materialized query tables that it can use to rewrite the query, it might use multiple tables simultaneously. If DB2 cannot use the tables simultaneously, it chooses which one to use according to a set of rules.

After writing the new query, DB2 determines the cost and the access path of that query. DB2 uses the rewritten query if the estimated cost of the rewritten query is less than the estimated cost of the original query. The rewritten query might give only approximate results if the data in the materialized query table is not up to date.

PSPI

Automatic query rewrite—complex examples

These examples can help you understand how DB2 applies automatic query rewrite to avoid costly aggregations and joins against large fact tables.

The following examples assume a scenario in which a data warehouse has a star schema. The star schema represents the data of a simplified credit card application, as shown in the following figure.

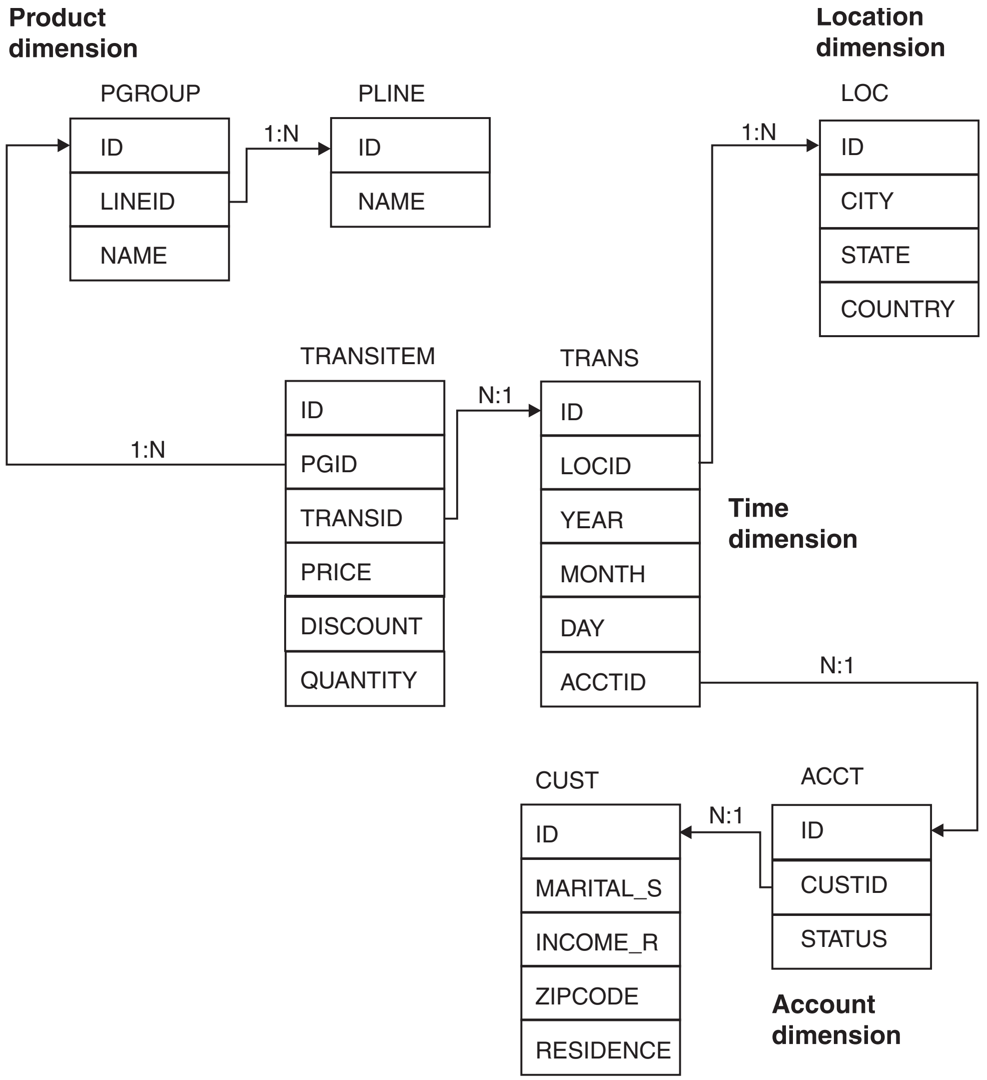


Figure 14. Multi-fact star schema. In this simplified credit card application, the fact tables TRANSITEM and TRANS form the hub of the star schema. The schema also contains four dimensions: product, location, account, and time.

The data warehouse records transactions that are made with credit cards. Each transaction consists of a set of items that are purchased together. At the center of the data warehouse are two large fact tables. TRANS records the set of credit card purchase transactions. TRANSITEM records the information about the items that are purchased. Together, these two fact tables are the hub of the star schema. The star schema is a multi-fact star schema because it contains these two fact tables. The fact tables are continuously updated for each new credit card transaction.

In addition to the two fact tables, the schema contains four dimensions that describe transactions: product, location, account, and time.

- The product dimension consists of two normalized tables, PGROUP and PLINE, that represent the product group and product line.
- The location dimension consists of a single, denormalized table, LOC, that contains city, state, and country.
- The account dimension consists of two normalized tables, ACCT and CUST, that represent the account and the customer.
- The time dimension consists of the TRANS table that contains day, month, and year.

Analysts of such a credit card application are often interested in the aggregation of the sales data. Their queries typically perform joins of one or more dimension tables with fact tables. The fact tables contain significantly more rows than the dimension tables, and complicated queries that involve large fact tables can be very costly. In many cases, you can use materialized query table to summarize and store information from the fact tables. Using materialized query tables can help you avoid costly aggregations and joins against large fact tables.

PSPI

Example 1

GUPI

An analyst submits the following query to count the number of transactions that are made in the United States for each credit card. The analyst requests the results grouped by credit card account, state, and year:

```
UserQ1
------
SELECT T.ACCTID, L.STATE, T.YEAR, COUNT(*) AS CNT
  FROM TRANS T, LOC L
  WHERE T.LOCID = L.ID AND
        L.COUNTRY = 'USA'
  GROUP BY T.ACCTID, L.STATE, T.YEAR;
```

Assume that the following CREATE TABLE statement created a materialized query table named TRANSCNT:

```
CREATE TABLE TRANSCNT AS
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) AS CNT
     FROM TRANS
     GROUP BY ACCTID, LOCID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED;
```

If you enable automatic query rewrite, DB2 can rewrite UserQ1 as NewQ1. NewQ1 accesses the TRANSCNT materialized query table instead of the TRANS fact table.

```
NewQ1
-----
SELECT A.ACCTID, L.STATE, A.YEAR, SUM(A.CNT) AS CNT
  FROM TRANSCNT A, LOC L
  WHERE A.LOCID = L.ID     AND
        L.COUNTRY = 'USA'
  GROUP BY A.ACCTID, L.STATE, A.YEAR;
```

DB2 can use query rewrite in this case because of the following reasons:

- The TRANS table is common to both UserQ1 and TRANSCNT.
- DB2 can derive the columns of the query result from TRANSCNT.
- The GROUP BY in the query requests data that are grouped at a higher level than the level in the definition of TRANSCNT.

Because customers typically make several hundred transactions per year with most of them in the same city, TRANSCNT is about hundred times smaller than TRANS. Therefore, rewriting UserQ1 into a query that uses TRANSCNT instead of TRANS improves response time significantly.

GUPI

Example 2

GUPI

Assume that an analyst wants to find the number of televisions, with a price over 100 and a discount greater than 0.1, that were purchased by each credit card account. The analyst submits the following query:

```
UserQ2
------
SELECT T.ID, TI.QUANTITY * TI.PRICE * (1 - TI.DISCOUNT) AS AMT
  FROM TRANSITEM TI, TRANS T, PGROUP PG
  WHERE TI.TRANSID = T.ID  AND
        TI.PGID = PG.ID    AND
        TI.PRICE > 100     AND
        TI.DISCOUNT > 0.1  AND
        PG.NAME = 'TV';
```

If you define the following materialized query table TRANSIAB, DB2 can rewrite UserQ2 as NewQ2:

```
TRANSIAB
--------
CREATE TABLE TRANSIAB AS
(SELECT TI.TRANSID, TI.PRICE, TI.DISCOUNT, TI.PGID,
        L.COUNTRY, TI.PRICE * TI.QUANTITY as VALUE
   FROM TRANSITEM TI, TRANS T, LOC L
   WHERE TI.TRANSID = T.ID  AND
         T.LOCID = L.ID     AND
         TI.PRICE > 1       AND
         TI.DISCOUNT > 0.1)
   DATA INITIALLY DEFERRED
   REFRESH DEFERRED;


NewQ2
-----
SELECT A.TRANSID, A.VALUE * (1 - A.DISCOUNT) as AM
  FROM TRANSIAB A, PGROUP PG
  WHERE A.PGID = PG.ID     AND
        A.PRICE > 100      AND
        PG.NAME = 'TV';
```

DB2 can rewrite UserQ2 as a new query that uses materialized query table TRANSIAB because of the following reasons:

- Although the predicate T.LOCID = L.ID appears only in the materialized query table, it does not result in rows that DB2 might discard. The referential constraint between the TRANS.LOCID and LOC.ID columns makes the join between TRANS and LOC in the materialized query table definition lossless. The join is *lossless* only if the foreign key in the constraint is NOT NULL.
- The predicates TI.TRANSID = T.ID and TI.DISCOUNT > 0.1 appear in both the user query and the TRANSIAB fullselect.
- The fullselect predicate TI.PRICE >1 in TRANSIAB subsumes the user query predicate TI.PRICE > 100 in UserQ2. Because the fullselect predicate is more inclusive than the user query predicate, DB2 can compute the user query predicate from TRANSIAB.
- The user query predicate PG.NAME = 'TV' refers to a table that is not in the TRANSIAB fullselect. However, DB2 can compute the predicate from the PGROUP table. A predicate like PG.NAME = 'TV' does not disqualify other

predicates in a query from qualifying for automatic query rewrite. In this case PGROUP is a relatively small dimension table, so a predicate that refers to the table is not overly costly.

- DB2 can derive the query result from the materialized query table definition, even when the derivation is not readily apparent:
  - DB2 derives T.ID in the query from T.TRANSID in the TRANSIAB fullselect. Although these two columns originate from different tables, they are equivalent because of the predicate `T.TRANSID = T.ID`. DB2 recognizes such column equivalency through join predicates. Thus, DB2 derives T.ID from T.TRANSID, and the query qualifies for automatic query rewrite.
  - DB2 derives AMT in the query UserQ2 from DISCOUNT and VALUE in the TRANSIAB fullselect.

GUPI

Example 3

GUPI

This example shows how DB2 matches GROUP BY items and aggregate functions between the user query and the materialized query table fullselect. Assume that an analyst submits the following query to find the average value of the transaction items for each year:

```
UserQ3
------
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
  FROM TRANSITEM TI, TRANS T
  WHERE TI.TRANSID = T.ID
  GROUP BY YEAR;
```

If you define the following materialized query table TRANSAVG, DB2 can rewrite UserQ3 as NewQ3:

```
TRANSAVG
--------
CREATE TABLE TRANSAVG AS
  (SELECT T.YEAR, T.MONTH, SUM(QUANTITY * PRICE) AS TOTVAL, COUNT(*) AS CNT
     FROM TRANSITEM TI, TRANS T
     WHERE TI.TRANSID = T.ID
     GROUP BY T.YEAR, T.MONTH )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED;
```

```
NewQ3
-----
SELECT YEAR, CASE WHEN SUM(CNT) = 0 THEN NULL
                  ELSE SUM(TOTVAL)/SUM(CNT)
                  END AS AVGVAL
  FROM TRANSAVG
  GROUP BY YEAR;
```

DB2 can rewrite UserQ3 as a new query that uses materialized query table TRANSAVG because of the following reasons:

- DB2 considers YEAR in the user query and YEAR in the materialized query table fullselect to match exactly.

- DB2 can derive the AVG function in the user query from the SUM function and the COUNT function in the materialized query table fullselect.
- The GROUP BY clause in the query NewQ3 requests data at a higher level than the level in the definition of TRANSAVG.
- DB2 can compute the yearly average in the user query by using the monthly sums and counts of transaction items in TRANSAVG. DB2 derives the yearly averages from the CNT and TOTVAL columns of the materialized query table by using a case expression.

GUPI

Determining whether query rewrite occurred

You can use EXPLAIN to determine whether DB2 has rewritten a user query to use a materialized query table.

About this task

PSPI

When DB2 rewrites the query, the PLAN TABLE shows the name of the materialized query that DB2 uses. The value of the TABLE_TYPE column is M to indicate that the table is a materialized query table.

Example

Consider the following user query:

```
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
  FROM TRANSITEM TI, TRANS T
  WHERE TI.TRANSID = T.ID
  GROUP BY YEAR;
```

If DB2 rewrites the query to use a materialized query table, a portion of the plan table output might look like the following table.

Table 53. Plan table output for an example with a materialized query table

PLANNO	METHOD	TNAME	JOIN_TYPE	TABLE_TYPE
1	0	TRANSAVG	-	M
2	3	2	-	?

The value M in TABLE_TYPE indicates that DB2 used a materialized query table. TNAME shows that DB2 used the materialized query table named TRANSAVG.You can also obtain this information from a performance trace (IFCID 0022).

PSPI

Enabling automatic query rewrite

Whether DB2 can consider automatic query rewrite depends on properly defined materialized query tables, and the values of two special registers.

Before you begin

- The isolation levels of the materialized query tables must be equal to or higher than the isolation level of the dynamic query being considered for automatic query rewrite
- You must populate system-maintained materialized query tables before DB2 considers them in automatic query rewrite.

About this task

GUPI

The values of two special registers, CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION determine whether DB2 can consider using materialized query tables in automatic query rewrite.

Procedure

To enable automatic query rewrite:

- Specify ANY for the CURRENT REFRESH AGE special register.

  The value in special register CURRENT REFRESH AGE represents a refresh age. The *refresh age* of a materialized query table is the time since the REFRESH TABLE statement last refreshed the table. (When you run the REFRESH TABLE statement, you update the timestamp in the REFRESH_TIME column in catalog table SYSVIEWS.) The special register CURRENT REFRESH AGE specifies the maximum refresh age that a materialized query table can have. Specifying the maximum age ensures that automatic query rewrite does not use materialized query tables with old data. The CURRENT REFRESH AGE has only two values: 0 or ANY. A value of 0 means that DB2 considers no materialized query tables in automatic query rewrite. A value of ANY means that DB2 considers all materialized query tables in automatic query rewrite.

  The CURRENT REFRESH AGE field on installation panel DSNTIP8 determines the initial value of the CURRENT REFRESH AGE special. The default value for the CURRENT REFRESH AGE field is 0.
- Specify the appropriate value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.

  The refresh age of a user-maintained materialized query table might not truly represent the freshness of the data in the table. In addition to the REFRESH TABLE statement, user-maintained query tables can be updated with the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements and the LOAD utility. Therefore, you can use the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register to determine which type of materialized query tables, system-maintained or user-maintained, DB2 considers in automatic query rewrite. The special register has four possible values that indicate which materialized query tables DB2 considers for automatic query rewrite:

  SYSTEM
  DB2 considers only system-maintained materialized query tables.

  USER DB2 considers only user-maintained materialized query tables.

  ALL DB2 considers both types of materialized query tables.

  NONE
  DB2 considers no materialized query tables.

The CURRENT MAINT TYPES field on installation panel DSNTIP4 determines the initial value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. the default value for CURRENT MAINT TYPES is SYSTEM.

Results

The following table summarizes how to use the CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers together. The table shows which materialized query tables DB2 considers in automatic query rewrite.

Table 54. The relationship between CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers

Value of CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	**SYSTEM**	**USER**	**ALL**	**None**
CURRENT REFRESH AGE=ANY	All system-maintained materialized query tables	All user-maintained materialized query tables	All materialized query tables (both system-maintained and user-maintained)	None
CURRENT REFRESH AGE=0	None	None	None	None

GUPI

Creating a materialized query table

You can create a materialized query table, which is defined by the result of a query, to improve the performance of certain SQL applications.

About this task

GUPI

You should create materialized query tables in a table space that is defined as NOT LOGGED to avoid the performance overhead created by logging changes to the data.

Procedure

To create a new table as a materialized query table:

1. Write a CREATE TABLE statement, and specify a fullselect. You can explicitly specify the column names of the materialized query table or allow DB2 to derive the column names from the fullselect. The column definitions of a materialized query table are the same as those for a declared global temporary table that is defined with the same fullselect.
2. Include the DATA INITIALLY DEFERRED and REFRESH DEFERRED clauses when you define a materialized query table.

   DATA INITIALLY DEFERRED clause

   DB2 does not populate the materialized query table when you create the table. You must explicitly populate the materialized query table.

- For system-maintained materialized query tables, populate the tables for the first time by using the REFRESH TABLE statement.
- For user-maintained materialized query tables, populate the table by using the LOAD utility, INSERT statement, or REFRESH TABLE statement.

REFRESH DEFERRED clause

DB2 does not immediately update the data in the materialized query table when its base tables are updated. You can use the REFRESH TABLE statement at any time to update materialized query tables and maintain data currency with underlying base tables.

3. Specify who maintains the materialized query table:

MAINTAINED BY SYSTEM clause

Specifies that the materialized query table is a system-maintained materialized query table. You cannot update a system-maintained materialized query table by using the LOAD utility or the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements. You can update a system-maintained materialized query table only by using the REFRESH TABLE statement. BY SYSTEM is the default behavior if you do not specify a MAINTAINED BY clause.

Create system-maintained materialized query tables in segmented or universal table spaces because the REFRESH TABLE statement triggers a mass delete operation.

MAINTAINED BY USER clause

Specifies that the table is a user-maintained materialized query table. You can update a user-maintained materialized query table by using the LOAD utility, the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements, as well as the REFRESH TABLE statement.

4. Specify whether query optimization is enabled.

ENABLE QUERY OPTIMIZATION clause

Specifies that DB2 can consider the materialized query table in automatic query rewrite. When you enable query optimization, DB2 is more restrictive of what you can select in the fullselect for a materialized query table.

DISABLE QUERY OPTIMIZATION clause

Specifies that DB2 cannot consider the materialized query table in automatic query rewrite.

Recommendation: When creating a user-maintained materialized query table, initially disable query optimization. Otherwise, DB2 might automatically rewrite queries to use the empty materialized query table. After you populate the user-maintained materialized query table, you can alter the table to enable query optimization.

Results

The isolation level of the materialized table is the isolation level at which the CREATE TABLE statement is executed.

After you create a materialized query table, it looks and behaves like other tables in the database system, with a few exceptions. DB2 allows materialized query tables in database operations wherever it allows other tables, with a few restrictions. As with any other table, you can create indexes on the materialized query table; however, the indexes that you create must not be unique. Instead, DB2 uses the materialized query table's definition to determine if it can treat the index as a unique index for query optimization.

Example

The following CREATE TABLE statement defines a materialized query table named TRANSCNT. The TRANSCNT table summarizes the number of transactions in table TRANS by account, location, and year:

```
CREATE TABLE TRANSCNT (ACCTID, LOCID, YEAR, CNT) AS
  (SELECT ACCOUNTID, LOCATIONID, YEAR, COUNT(*)
     FROM TRANS
     GROUP BY ACCOUNTID, LOCATIONID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED
     MAINTAINED BY SYSTEM
     ENABLE QUERY OPTIMIZATION;
```

The fullselect, together with the DATA INITIALLY DEFERRED clause and the REFRESH DEFERRED clause, defines the table as a materialized query table.

GUPI

Related reference:

CREATE TABLE (DB2 SQL)

Rules for materialized query table

If one or more source tables in the materialized query table definition contain a security label column, certain rules apply to creating a materialized query table.

About this task

GUPI

- If only one source table contains a security label column, the following conditions apply:
  - You must define the security label column in the materialized query table definition with the AS SECURITY LABEL clause.
  - The materialized query table inherits the security label column from the source table.
  - The MAINTAINED BY USER option is allowed.
- If only one source table contains a security label column and the materialized query table is defined with the DEFINITION ONLY clause, the materialized query table inherits the values in the security label column from the source table. However, the inherited column is not a security label column.
- If more than one source table contains a security label column, DB2 returns an error code and the materialized query table is not created.

GUPI

Registering an existing table as a materialized query table

You might already have manually created base tables that act like materialized query tables and have queries that directly access the base tables. These base tables are often referred to as *summary tables*.

Before you begin

To ensure the accuracy of data that is used in automatic query rewrite, ensure that the summary table is current before registering it as a materialized query table.

Alternatively, you can follow these steps:

1. Register the summary table as a materialized query table with automatic query rewrite disabled.
2. Update the newly registered materialized query table to refresh the data.
3. Use the ALTER TABLE statement on the materialized query table to enable automatic query rewrite.

Procedure

To take advantage of automatic query rewrite for an existing summary table:

Use the ALTER TABLE statement with DATA INITIALLY DEFERRED and MAINTAINED BY USER clauses, to register the table as materialized query table. The fullselect must specify the same number of columns as the table you register as a materialized query table. The columns must have the same definitions and have the same column names in the same ordinal positions.
The DATA INITIALLY DEFERRED clause indicates that the table data is to remain the same when the ALTER statement completes. The MAINTAINED BY USER clause indicates that the table is user-maintained.

Results

The table becomes immediately eligible for use in automatic query rewrite. The isolation level of the materialized query table is the isolation level at which the ALTER TABLE statement is executed.

You can continue to update the data in the table by using the LOAD utility or the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements. You can also use the REFRESH TABLE statement to update the data in the table.

Example

GUPI

Assume that you have an existing summary table named TRANSCOUNT. The TRANSCOUNT tabl has four columns to track the number of transactions by account, location, and year. Assume that TRANSCOUNT was created with this CREATE TABLE statement:

```
CREATE TABLE TRANSCOUNT
   (ACCTID    INTEGER NOT NULL
    LOCID     INTEGER NOT NULL
    YEAR      INTEGER NOT NULL
    CNT       INTEGER NOT NULL);
```

The following SELECT statement then populated TRANSCOUNT with data that was derived from aggregating values in the TRANS table:

```
SELECT ACCTID, LOCID, YEAR, COUNT(*)
FROM TRANS
GROUP BY ACCTID, LOCID, YEAR ;
```

You could use the following ALTER TABLE statement to register TRANSCOUNT as a materialized query table. The statement specifies the ADD MATERIALIZED QUERY clause:

```
ALTER TABLE TRANSCOUNT ADD MATERIALIZED QUERY
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) as cnt
     FROM TRANS
     GROUP BY ACCTID, LOCID, YEAR )
     DATA INITIALLY DEFERRED
     REFRESH DEFERRED
     MAINTAINED BY USER;
```

GUPI

Related reference:

ALTER TABLE (DB2 SQL)

Altering an existing materialized query table

You can use the ALTER TABLE statement to change the attributes of a materialized query table or change a materialized query table to a base table.

About this task

Altering a materialized query table to enable it for query optimization makes the table immediately eligible for use in automatic query rewrite. You must ensure that the data in the materialized query table is current. Otherwise, automatic query rewrite might return results that are not current.

One reason you might want to change a materialized query table into a base table is to perform table operations that are restricted for a materialized query table. For example, you might want to rotate the partitions on your partitioned materialized query table. In order to rotate the partitions, you must change your materialized query table into a base table. While the table is a base table, you can rotate the partitions. After you rotate the partitions, you can change the table back to a materialized query table.

In addition to using the ALTER TABLE statement, you can change a materialized query table by dropping the table and recreating the materialized query table with a different definition.

Procedure

To change the attributes of a materialized query table:

1. Issue an ALTER TABLE statement.
   - Enable or disable automatic query rewrite for a materialized query table with the ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION clause.
   - Change the type of materialized query table between system-maintained and user-maintained by using the MAINTAINED BY SYSTEM or MAINTAINED BY USER clause.
   -
2. Change a materialized query table into a base table.

   GUPI

For example Assume that you no longer want the TRANSCOUNT table to be a materialized query table. The following ALTER TABLE statement, which specifies the DROP MATERIALIZED QUERY clause, changes the materialized query table into a base table.

```
ALTER TABLE TRANSCOUNT DROP MATERIALIZED QUERY;
```

GUPI

The column definitions and the data in the table do not change. However, DB2 can no longer use the table in automatic query rewrite. You can no longer update the table with the REFRESH TABLE statement.

Populating and maintaining materialized query tables

After you define a materialized query table, you need to maintain the accuracy of the data in the table.

About this task

This maintenance includes populating the table for the first time and periodically refreshing the data in the table. You need to refresh the data because any changes that are made to the base tables are not automatically reflected in the materialized query table.

The only way to change the data in a system-maintained materialized query table is through the REFRESH TABLE statement. The INSERT, UPDATE, MERGE, TRUNCATE and DELETE statements, and the LOAD utility cannot refer to a system-maintained materialized query table as a target table. Therefore, a system-maintained materialized query table is read-only.

Any view or cursor that is defined on a system-maintained materialized query table is read-only. However, for a user-maintained materialized query table, you can alter the data with the INSERT, UPDATE, and DELETE statements, and the LOAD utility.

PSPI

Populating a new materialized query table

You can populate a newly created table for the first time by using the REFRESH TABLE statement or by using INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility.

About this task

When you create a materialized query table with the CREATE TABLE statement, the table is not immediately populated.

Procedure

To populate a materialized query table:

- Issue a REFRESH TABLE statement.

  GUPI

  For example, the following REFRESH TABLE statement populates a materialized query table named SALESCNT:

```
REFRESH TABLE SALESCNT;
```

GUPI

You should avoid using the REFRESH TABLE statement to update user-maintained materialize query tables because the REFRESH TABLE statement uses a fullselect and can result in a long-running query.

The REFRESH TABLE statement is an explainable statement. The explain output contains rows for INSERT with the fullselect in the materialized query table definition.

- Use the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility.

  You cannot use the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility to change system-maintained materialized query tables.

Refreshing a system-maintained materialized query table

You can use the REFRESH TABLE statement to refresh the data in any materialized query table at any time.

Procedure

To refresh an existing materialized query table:

Issue a REFRESH TABLE statement. When you issue the REFRESH TABLE statement DB2performs the following actions:

1. Deletes all the rows in the materialized query table
2. Executes the fullselect in the materialized query table definition to recalculate the data from the tables that are specified in the fullselect with the isolation level for the materialized query table
3. Inserts the calculated result into the materialized query table
4. Updates the DB2 catalog with a refresh timestamp and the cardinality of the materialized query table

Although the REFRESH TABLE statement involves both deleting and inserting data, DB2 completes these operations in a single commit scope. Therefore, if a failure occurs during execution of the REFRESH TABLE statement, DB2 rolls back all changes that the statement made.

Refreshing user-maintained materialized query tables

You can update the data in user-maintained materialized query tables by using the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements, and the LOAD utility.

About this task

PSPI

You should avoid using the REFRESH TABLE statement to update user-maintained materialize query tables. Because the REFRESH TABLE statement uses a fullselect to refresh a materialized query table, the statement can result in a long-running query. Use insert, update, delete, or load operations might be more efficient than using the REFRESH TABLE statement.

Depending on the size and frequency of changes in base tables, you might use different strategies to refresh your materialized query tables. For example, for infrequent, minor changes to dimension tables, you could immediately propagate the changes to the materialized query tables by using triggers. For larger or more frequent changes, you might consider refreshing your user-maintained materialized query tables incrementally to improve performance.

Procedure

To avoid refresh a user-maintained materialized query table:

Use INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility. For example, you might find it faster to generate the data for your materialized query table and execute the LOAD utility to populate the data.

Example

For example, assume that you need to add a large amount of data to a fact table. Then, you need to refresh your materialized query table to reflect the new data in the fact table. To do this, perform these steps:

- Collect and stage the new data in a separate table.
- Evaluate the new data and apply it to the materialized table as necessary.
- Merge the new data into the fact table

For an example of such code, see member DSNTEJ3M in DSN1010.SDSNSAMP, which is shipped with DB2.

PSPI

Updating statistics on materialized query tables

For optimal performance of materialized query tables, you need to provide DB2 with accurate catalog statistics for access path selection.

About this task

PSPI

When you run the REFRESH TABLE statement, the only statistic that DB2 updates for the materialized query table is the cardinality statistic.

Procedure

To keep catalog statistics current for materialized query tables:

Run the RUNSTATS utility after executing a REFRESH TABLE statement or after changing the materialized query table significantly. Otherwise, DB2 uses default or out-of-date statistics. The estimated performance of queries that are generated by automatic rewrite might inaccurately compare less favorably to the original query.

PSPI

Rules for using materialized query tables in a multilevel security environment

If source tables have multilevel security with row-level granularity enabled, some additional rules apply to working with the materialized query table and the source tables.

GUPI

Tables with multilevel security enabled contain a security label column, which is defined with the AS SECURITY LABEL clause. The values in the security label column indicate which users can access the data in each row.

Creating a materialized query tablet

If one or more source tables in the materialized query table definition contain a security label column, certain rules apply to creating a materialized query table.

Only one source table contains a security label column

The following conditions apply:

- You must define the security label column in the materialized query table definition with the AS SECURITY LABEL clause.
- The materialized query table inherits the security label column from the source table.
- The MAINTAINED BY USER option is allowed.

Only one source table contains a security label column, and a DEFINITION ONLY clause was used

The materialized query table inherits the values in the security label column from the source table. However, the inherited column is not a security label column.

More than one source table contains a security label column

DB2 returns an error code, and the materialized query table is not created.

Altering a source table

An ALTER TABLE statement to add a security label column to a table fails if the table is a source table for a materialized query table.

Refreshing a materialized query table

The REFRESH TABLE statement deletes the data in a materialized query table and then repopulates the materialized query table according to its table definition. During this refresh process, DB2 does not check for multilevel security with row-level granularity.

GUPI

Related concepts:

Multilevel security (Managing Security)

Enabling a materialized query table for automatic query rewrite

After you populate a user-maintained materialized query table, you can alter the table to enable query optimization.

Before you begin

If the materialized query table is user-maintained, it is populated with data.

About this task

PSPI

When creating a user-maintained materialized query table, initially disable query optimization. Otherwise, DB2 might automatically rewrite queries to use the empty materialized query table.

Procedure

To enable a materialized query table for automatic query rewrite:

Issue an ALTER TABLE statement and specify:
- The ENABLE QUERY OPTIMIZATION clause
- An isolation level equivalent or higher than that of the dynamic queries that might use the materialized query table. The isolation level of the table must be equal to or higher than the isolation level of the dynamic query being considered for automatic query rewrite.

PSPI

Recommendations for materialized query table and base table design

By following certain best practices, you might improve the performance of your materialized query tables and the queries that use them. These recommendations, however, do not represent a complete design theory.

Designing materialized query tables for automatic query rewrite

By following these recommendations, you might improve the performance of queries that use materialized query tables.

Procedure

GUPI

To get better performance from your materialized query tables:
- Include aggregate functions strategically in the fullselect of a materialized query table definition:
  - Include COUNT(*) and SUM(*expression*).
  - Include SUM(*expression*expression*) only if you plan to query VAR(*expression*), STDDEV(expression), VAR_SAMP(expression), or STDDEV_SAMP(*expression*).
  - Include COUNT(*expression*) in addition to COUNT(*) if expression is nullable.
  - Include MIN(*expression*) and MAX(*expression*) if you plan to query them.
  - Do not include AVG(*expression*), VAR(*expression*), or STDDEV(*expression*) directly if you include either of the following parameter combinations:
    - SUM(*expression*), SUM(*expression*expression*), and COUNT(*)
    - SUM(*expression*), SUM(*expression*expression*), and COUNT(expression)

DB2 can derive AVG(*expression*), VAR(*expression*), and STDDEV(*expression*) from SUM(*expression*), SUM(*expression**expression*), and the appropriate COUNT aggregate function.

- Include the foreign key of a dimension table in the GROUP BY clause of a materialized query table definition. For example, if you include PGROUP.ID, also include PGROUP.LINEID. Then DB2 can use the materialized query table to derive a summary at the LINEID level, without rejoining PGROUP.
- Include all the higher-level columns in the materialized query table if DB2 does not know the functional dependency in a denormalized dimension table. For example, if you include CITY in a GROUP BY clause, also include STATE and COUNTRY in the clause. Similarly, if you include MONTH in the GROUP BY clause, also include YEAR in the clause.
- Do not use the HAVING clause in your materialized query tables. A materialized query table with a HAVING clause in its definition has limited usability during query rewrite.
- Create indexes on materialized query tables as you would for base tables.

GUPI

Designing base tables for automatic query rewrite

These recommendations describe base table design strategies that might increase the performance and eligibility of your materialized query tables.

Procedure

GUPI

To make your base tables work well with materialized query tables:

- Define referential integrity as either ENFORCED or NOT ENFORCED whenever possible.
- Define an index as unique if it is truly unique.
- Define all base table columns as NOT NULL if possible, so that COUNT(*x*) is the same as COUNT(*). Then you do not need to include COUNT(*x*) for each nullable column *x* in a materialized query table definition. If necessary, use a special value to replace NULL.
- Emphasize normalized dimension design over denormalized dimension design in your base tables. When you use normalized dimension design, you do not need to include non-primary key columns in a materialized query table, thereby saving you storage space. DB2 compensates for the lack of non-primary key columns by deriving these columns through a re-join of the dimension table. If normalization causes performance problems, you can define materialized query tables to denormalize the snowflake dimensions.

Materialized query tables—examples shipped with DB2

In addition to the examples shown in this information, DB2 provides a number of samples to help you design materialized query tables for automatic query rewrite.

PSPI

The samples are based on a data warehouse with a star schema database. The star schema contains one fact table, SALESFACT, and these four hierarchical dimensions:

- A TIME dimension that consists of one dimension table
- A PRODUCT dimension that is a snowflake that consists of four fully normalized tables
- A LOCATION dimension that is a snowflake that consists of five partially normalized tables
- A CUSTOMER dimension that is a snowflake that consists of four fully normalized tables

See member DSNTEJ3M in data set DSN810.SDSNSAMP for all of the code, including the following items:

- SQL statements to create and populate the star schema
- SQL statements to create and populate the materialized query tables
- Queries that DB2 rewrites to use the materialized query table

PSPI

Chapter 22. Improving performance for LOB data

You can improve the performance of applications that access LOB data by specifying that an inline portion of LOB data columns be stored in the base table space along with data from the other non-LOB columns.

About this task

PSPI

An *inline LOB* allows a portion of LOB data to reside in the base table space with the data from non-LOB columns. For LOBs of a size less than or equal to the specified inline length, DB2 stores the complete LOB data in the base table space. DB2 does not need to access the LOB table space or auxiliary indexes for processes that access the LOB data.

In such cases, DB2 can access the data at similar cost, in terms of CPU time and elapsed time, for comparable non-LOB data types. The amount of disk space that is used for LOB data is also reduced when the LOB data can be stored inline in the base table space.

For LOBs of a size greater than the specified inline length, the inline portion of the LOB resides in the base table space, and DB2 stores the remainder of the LOB in the LOB table space. In this case, any process that accesses the LOB data must access both the base table space and the LOB table space.

The benefits of inline LOBs are greatest for frequently accessed LOB columns. For LOB column that are accessed only rarely, the presence of the inline LOB data in the base table might reduce the number of row-per-page enough to incur increased I/O costs that outweigh any benefits of keeping the LOB data inline.

Procedure

To specify a length for inline LOBs, use the following approaches:

- Use the LOB_INLINE_LENGTH subsystem parameter to specify a default inline length for any new LOB column in universal table spaces on the DB2 subsystem. Valid values for the LOB_INLINE_LENGTH subsystem parameter are 0 to 32680 inclusive, in bytes. The default value is 0, which means no inline attribute is used for LOB columns. A non-zero value specifies that new LOB columns created on the subsystem have an inline attribute, and the value indicates how many bytes of data DB2 stores in the base table space with data from non-LOB columns. For example, assuming that 1001 is specified for the value of the LOB_INLINE_LENGTH parameter:
  - If the length of the LOB data is 200 bytes, DB2 stores all 200 bytes in the base table space.
  - If the length of the LOB is 2000 bytes, DB2 stores 1001 bytes in the base table space, and 999 bytes in the LOB table space.

  DB2 interprets the value specified for the LOB_INLINE_LENGTH parameter in bytes regardless of the data type or sub-type of the LOB column. If an odd number is specified for this parameter, DB2 rounds the value up to the next even number for any DBCLOB column.

- Specify the INLINE LENGTH clause of a CREATE TYPE statement. Any LOB-based column in a universal table space can inherit the inline attribute from the distinct type. You can specify a value from 0 to 32680 bytes inclusive for types based on BLOB or CLOB, and 0 to 16340 characters inclusive for types base on DBCLOB.
- Specify the INLINE LENGTH clause of a CREATE TABLE or ALTER TABLE ADD statement for a table in a universal table space. You can specify a value from 0 to 32680 inclusive for BLOB and CLOB columns, and from 0 to 16340 inclusive for DBCLOB columns. For example, consider the columns created by the following statements:

  ```
  CREATE TABLE myLOBtable
  (myLOBcolumn DBCLOB (500K) INLINE LENGTH 300);
  ```

  If the actual length of the LOB is 200 bytes (100 characters) all 200 bytes are stored in the base table space. If the length of the LOB is 2000 bytes (1000 characters), 600 bytes (300 characters) are stored in the base table space and 1400 bytes (700 characters) are stored in the LOB table space.

  PSPI

Related concepts:

Optimization of inline LOBs (DB2 for z/OS What's New?)

Compression after materialization of inline LOB changes (DB2 Utilities)

Related tasks:

Choosing data page sizes for LOB data

Related reference:

LOB INLINE LENGTH field (LOB_INLINE_LENGTH subsystem parameter) (DB2 Installation and Migration)

CREATE TABLE (DB2 SQL)

ALTER TABLE (DB2 SQL)

CREATE TYPE (distinct) (DB2 SQL)

Chapter 23. Choosing data page sizes for LOB data

Choosing a data page size for LOBs (in the LOB table space) is a trade-off between minimizing the number of getpage operations (maximizing performance) and not wasting space.

About this task

With LOB table spaces, no more than one LOB value is ever stored in any single data page in a LOB table space. Space that is not used by the LOB value in the last page that is occupied by the LOB remains unused. DB2 also uses additional space for control information. The smaller the LOB value, the greater the proportion of space for this “non-data” is used.

For example, if you have a 17-KB LOB, the 4-KB page size is the most efficient for storage. A 17-KB LOB requires five 4-KB pages for a total of 20 KB of storage space. Pages that are 8 KB, 16 KB, and 32 KB in size waste more space, because they require 24 KB, 32 KB, and 32 KB, respectively, for the LOB.

Procedure

To optimize the use of space by LOB data, use the following approaches:

- If not all LOB values are the same size, use the following formula to estimate the size:

  ```
  LOB size = (average LOB length)
  × 1.05
  ```

  The following table contains some suggested page sizes for LOBs with the intent to reduce the number of I/O operations (getpages).

Table 55. Suggested page sizes based on average LOB length

Average LOB size (n)	Suggested page size
n ≤4 KB	4 KB
4 KB < n ≤ 8 KB	8 KB
8 KB < n ≤ 16 KB	16 KB
16 KB < n	32 KB

- If all LOB values are the same size, use the values in the following table to choose an appropriate page size:

Table 56. Suggested page sizes when LOBs are the same size

LOB size (y)	Suggested page size
y ≤ 4 KB	4 KB
4 KB < y ≤ 8 KB	8 KB
8 KB < y ≤ 12 KB	4 KB
12 KB < y ≤ 16 KB	16 KB
16 KB < y ≤ 24 KB	8 KB
24 KB < y ≤ 32 KB	32 KB
32 KB < y ≤ 48 KB	16 KB

Table 56. Suggested page sizes when LOBs are the same size (continued)

LOB size (y)	Suggested page size
48 KB < y	32 KB

Related tasks:

Creating table spaces explicitly (DB2 Administration Guide)

Choosing data page sizes

Choosing index page sizes

Chapter 22, “Improving performance for LOB data,” on page 273

Related information:

Implementing DB2 table spaces (DB2 Administration Guide)

Chapter 24. Reserving free space for table spaces

By reserving free space in table spaces you can enable your data to remain clustered longer between reorganizations and you can reduce the number of indirect references and overflow records, which can harm performance.

Before you begin

GUPI

To determine the amount of free space that is currently available on a page, run the RUNSTATS utility and examine the PERCACTIVE column of the SYSIBM.SYSTABLEPART catalog table.

About this task

When insufficient free space is available for insert or update operations, DB2 often appends new rows at the end of the table, out of clustering sequence. When updates to existing rows mean that they cannot fit on the original page, DB2 creates indirect references to overflow records on different data pages. When many of these records are physically located out of sequence, performance suffers.

Consequently, a sufficient amount of free space can provide the following advantages during normal processing:

- Data rows can remain clustered longer after data is reorganized or loaded because random inserts are not needed. Good clustering can improve buffer hit ratios and enables more use of dynamic prefetch. Without clustering, queries must rely on list prefetch for good performance. Data clustering might be less important if your storage hardware uses newer control units and solid state disks that can process list prefetch efficiently.
- Indirect references can be avoided. *Indirect references* are created when an update operation increases the size of a row so that it cannot fit on the original page that contained the row. DB2 stores the row in an overflow page and the original RID points to the overflow RID. Indirect references only occur for varying length rows. The most common cause of indirect references is the use of nullable VARCHAR columns, which initially contain null values and are later updated with non-null values. Indirect references are particularly problematic for queries that use random access or list prefetch. They cause additional CPU cost and more synchronous I/Os. DB2 cannot use list prefetch to read overflow records. You can monitor the NEARINDREF and FARINDREF columns in the SYSIBM.SYSTABLEPART catalog table to find how many rows have indirect references.
- Fewer data rows are locked by a single page lock, reducing contention when page-level locking is used.

However, specifying too much free space also has disadvantages, including:

- More disk space is used for the same amount of data.
- Less information can be transferred by a single I/O operation
- The same amount of data occupies more pages that must be scanned.
- Buffer pools and storage controller cache cannot be used as efficiently

You might not need to reserve any free space in certain situations, including:

- For read-only objects. If you do not plan to insert or update data in a table, no free space is needed for the table space.
- The object is not read-only, but inserts are at the end, and updates that lengthen varying-length columns are few.

Procedure

To improve the use of free space, use the following approaches:

- Use the PCTFREE clause in most situations. The PCTFREE clause specifies the percentage of each data page in a table space that is left free when loading or reorganizing the data.

  For example, the default value for table spaces PCTFREE 5, which means that 5% of each data page is kept free when you load or reorganize the data. DB2 reserves the specified amount of free space when data is loaded into the table or reorganized by utility operations. DB2 uses the free space later when you insert or update the data. Regardless of the specified PCTFREE values, at least one row is always inserted on each data page.

  The value of PCTFREE applies to the table spaces for hash-organized tables only when you invoke the REORG TABLESPACE utility and specify the AUTOESTSPACE(YES) option.
- If update activity on compressed data, which often results in longer rows, is heavy or insert volume is heavy, use a PCTFREE value greater than the default value.
- If you know the number of rows that fit on a data page at their maximum size, use the MAXROWS clause to control the number of rows per page. However, a MAXROWS value that is too small is likely to waste disk space, and a MAXROWS value that is too large is unlikely to prevent indirect references. The MAXROWS clause has the advantage of maintaining the free space even when new data is inserted.
- Use the FREEPAGE clause to specify how often DB2 leaves a full page of free space when loading data or when reorganizing data. For example, if you specify FREEPAGE 10, DB2 leaves every tenth page free. Use of the FREEPAGE clause is most appropriate in the following situations:
  - If MAXROWS is 1 or rows are larger than half a page. The PCTFREE clause has no impact in this case because DB2 cannot insert a second row on a page.
  - If additional free space is needed for catalog table spaces and indexes. The recommendation is to always use the default PCTFREE values for catalog objects.
- You might use MAXROWS values to improve concurrency for small tables and shared table spaces that use page-level locking. This approach reduces the number of rows per page, which helps to avoid lock contention. However, the use of row-level locking is an alternative to this approach and might be preferred in many situations.

Related tasks:

Increasing free space for compressed data

Maintaining data organization

Reserving free spaces for indexes

Related reference:

ALTER TABLESPACE (DB2 SQL)

CREATE TABLESPACE (DB2 SQL)

- SYSIBM.SYSTABLEPART table (DB2 SQL)
- REORG TABLESPACE (DB2 Utilities)

Related information:

- DB2 for z/OS and List Prefetch Optimizer (IBM Redbooks)

Chapter 25. Compressing your data

You can reduce the space required for a table by using data compression. Compressing the data in a table space can significantly reduce the amount of disk space that is needed to store data and can help improve buffer pool performance.

Before you begin

You can use the DSN1COMP utility to determine how well compression of your data will work. Data in a LOB table space or a table space that is defined in the work file database (the table space for declared temporary tables) cannot be compressed. For more information, see DSN1COMP (DB2 Utilities).

The CPU cost of both compression and decompression increases with smaller compression ratios. Therefore, it is best to avoid the use of compression if the compression ratio, or the percentage of saved space due to compression, is less than 10 - 20 percent. The additional CPU cost for compression and decompression makes it not worthwhile.

About this task

When you compress data, bit strings that occur frequently are replaced by shorter strings. Information about the mapping of bit strings to their replacements is stored in a *compression dictionary*. Computer processing is required to compress data before it is stored and to decompress the data when it is retrieved from storage. In many cases, using the COMPRESS clause can significantly reduce the amount of disk space needed to store data, but the compression ratio that you achieve depends on the characteristics of your data.

With compressed data, you might see some of the following performance benefits, depending on the SQL workload and the amount of compression:
- Higher buffer pool hit ratios
- Fewer I/Os
- Fewer getpage operations

Procedure

To compress data:
1. Specify COMPRESS YES in one of the following SQL statements:
   - CREATE TABLESPACE
   - ALTER TABLESPACE
2. Populate the table space with data by taking one of the following actions:
   - Run the LOAD utility with REPLACE, RESUME NO, or RESUME YES SHRLEVEL CHANGE, and without KEEPDICTIONARY.
   - Run the REORG utility without KEEPDICTIONARY.
   - Issue INSERT statements.
   - Issue MERGE statements.

   If no compression dictionary already exists, and the amount of data in the tables space reaches a threshold determined by DB2, a compression dictionary is created. After the compression dictionary is built, DB2 uses it to compress all subsequent data added to the table space.

Related tasks:

Compressing indexes

Compressing data by using the LOAD utility (DB2 Utilities)

Related reference:

LOAD (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

SELECT (DB2 SQL)

MERGE (DB2 SQL)

Deciding whether to compress data

You should consider many factors before you decide whether to compress data.

Consider these factors before compressing data:

Data row size

DB2 compresses the data of one record at a time. (The prefix of the record is not compressed.) As row lengths become shorter, compression yields diminishing returns because 8 bytes of overhead are required to store each record in a data page. On the other hand, when row lengths are very long, compression of the data portion of the row might yield little or no reduction in data set size because DB2 rows cannot span data pages. In the case of very long rows, using a larger page size can enhance the benefits of compression, especially if the data is accessed primarily in a sequential mode.

If compressing the record produces a result that is no shorter than the original, DB2 does not compress the record.

Table space size

Compression can work very well for large table spaces. With small table spaces, the size of the compression dictionary (64 KB) can offset the space savings that compression provides.

Processing costs

Decompressing a row of data costs significantly less than compressing that same row. The access path that DB2 chooses impacts the processor cost for data compression. In general, the relative overhead of compression is higher for table space scans and is less costlier for index access.

I/O costs

When rows are accessed sequentially, fewer I/Os might be required to access data that is stored in a compressed table space. However, the reduced I/O resource consumption is traded for extra processor cost for decoding the data.

- If random I/O is necessary to access the data, the number of I/Os does not decrease significantly, unless the associated buffer pool is larger than the table and the other applications require little concurrent buffer pool usage.

- Some types of data compress better than others. Data that contains hexadecimal characters or strings that occur with high frequency compresses quite well, while data that contains random byte frequencies might not compress at all. For example, textual and decimal data tends to compress well because certain byte strings occur frequently.

Data patterns

The frequency of patterns in the data determines the compression savings. Data with many repeated strings (such as state and city names or numbers with sequences of zeros) results in good compression savings.

Table space design

Each table space or partition that contains compressed data has a compression dictionary. The compression dictionary is built when you populate the table space with data.

The dictionary contains a fixed number of entries, usually 4096, and resides with the data. The dictionary content is based on the data at the time it was built, and does not change unless the dictionary is rebuilt or recovered, or compression is disabled with ALTER TABLESPACE.

If you use the REORG utility to build the compression dictionary, DB2 uses a sampling technique to build the dictionary. This technique uses the first *n* rows from the table space and then continues to sample rows for the remainder of the UNLOAD phase. The value of *n* is determined by how much your data can be compressed. In most cases, this sampling technique produces a better dictionary and might produce better results for table spaces that contain tables with dissimilar kinds of data.

Otherwise, DB2 uses only the first *n* rows added to the table space to build the contents of the dictionary.

If you have a table space that contains more than one table, and the data used to build the dictionary comes from only one or a few of those tables, the data compression might not be optimal for the remaining tables. Therefore, put a table that you want to compress into a table space by itself, or into a table space that only contains tables with similar kinds of data.

Existing exit routines

An exit routine is executed before compressing or after decompressing, so you can use DB2 data compression with your existing exit routines. However, do not use DB2 data compression in conjunction with DSN8HUFF. (DSN8HUFF is a sample edit routine that compresses data using the Huffman algorithm, which is provided with DB2. This adds little additional compression at the cost of significant extra CPU processing.

Logging effects

If a data row is compressed, all data that is logged because of SQL changes to that data is compressed. Thus, you can expect less logging for insertions and deletions; the amount of logging for updates varies. Applications that are sensitive to log-related resources can experience some benefit with compressed data.

External routines that read the DB2 log cannot interpret compressed data without access to the compression dictionary that was in effect when the data was compressed. However, using IFCID 306, you can cause DB2 to read log records of compressed data in decompressed format. You can retrieve those decompressed records by using the IFI function READS.

Distributed data
DB2 decompresses data before transmitting it to VTAM.

Related concepts:

The effect of data compression on performance (Introduction to DB2 for z/OS)

Huffman compression exit routine (DB2 Installation and Migration)

Related tasks:

Compressing your data

Compressing indexes

Reading complete log data (IFCID 0306) (DB2 Administration Guide)

Calculating the space that is required for a dictionary

A *dictionary* contains the information that is used for compressing and decompressing the data in a table space or partition. The dictionary resides in that same table space or partition.

About this task

If you are not going to compress data, you do not need to calculate the space that is required for a dictionary. Space allocation parameters are specified in pages (either 4 KB, 8 KB, 16 KB, or 32 KB).

Complete the following steps to calculate the disk space that is required by a dictionary, and the virtual storage that is required in the xxxxDBM1 address space when a dictionary is read into storage from a buffer pool.

Related tasks:

Estimating storage when using the LOAD utility (DB2 Administration Guide)

Calculating disk requirements for a dictionary

You can calculate the disk requirements for a dictionary that is associated with a compressed, nonsegmented table space, or a compressed, segmented table space.

Procedure

To calculate the disk requirements for a dictionary:

Determine your table space type:

Nonsegmented table space
The dictionary contains 4096 entries in most cases. Compression dictionaries require additional sixteen 4-KB pages, eight 8-KB pages, four 16-KB pages, or two 32-KB pages. Although it is possible that your dictionary can contain fewer entries, allocate enough space to accommodate a dictionary with 4096 entries.

Segmented table space
For segmented table spaces, additional space might be requiremed based on segment size and page size.

What to do next

Now, determine the virtual storage size that is required for a dictionary.

Calculating virtual storage requirements for a dictionary

Estimate the virtual storage as part of your calculations for the space required for a dictionary.

Procedure

To calculate how much storage is needed in the xxxxDBM1 address space for each dictionary:

Calculate the necessary dictionary size by using the following formula:

```
dictionary size (number of entries)
× 16 bytes
```

What to do next

When a dictionary is read into storage from a buffer pool, the *whole* dictionary is read, and it remains there as long as the compressed table space is being accessed.

Increasing free space for compressed data

You can provide free space to avoid the potential problem of more getpage and lock requests for compressed data.

About this task

In some cases, using compressed data results in an increase in the number of getpages, lock requests, and synchronous read I/Os. Sometimes, updated compressed rows cannot fit in the home page, and they must be stored in the overflow page. This can cause additional getpage and lock requests. If a page contains compressed fixed-length rows with no free space, an updated row probably has to be stored in the overflow page.

Procedure

To avoid the potential problem of more getpage and lock requests:

Add more free space within the page. Start with a PCTFREE 10 value to create 10% additional free space and adjust further, as needed. If, for example, 10% free space was used without compression, start with 20% free space with compression for most cases. A sufficient amount of free space is especially important for data that is heavily updated.

Related tasks:

Reserving free space for table spaces

Reserving free spaces for indexes

Related reference:

ALTER TABLESPACE (DB2 SQL)

CREATE TABLESPACE (DB2 SQL)

Determining the effectiveness of compression

Before compressing data, you can use the DSN1COMP stand-alone utility to estimate how well the data can be compressed.

About this task

After data is compressed, you can use compression reports and catalog statistics to determine how effectively it was compressed.

Procedure

To find the effectiveness of data compression:

- Use the DSN1COMP stand-alone utility to find out how much space can be saved and how much processing the compression of your data requires. Run DSN1COMP on a data set that contains a table space, a table space partition, or an image copy. DSN1COMP generates a report of compression statistics but does not compress the data.
- Examine the compression reports after you use REORG or LOAD to build the compression dictionary and compress the data. Both utilities issue a report message (DSNU234I or DSNU244I). The report message gives information about how well the data is compressed and how much space is saved. (REORG with the KEEPDICTIONARY option does not produce the report.)
- Query catalog tables to find information about data compression
  - PAGESAVE column of the SYSIBM.SYSTABLEPART tells you the percentage of pages that are saved by compressing the data.
  - PCTROWCOMP columns of SYSIBM.SYSTABLES and SYSIBM.SYSTABSTATS tells you the percentage of the rows that were compressed in the table or partition the last time RUNSTATS was run. Use the RUNSTATS utility to update these catalog columns.

Related reference:

DSN1COMP (DB2 Utilities)

RUNSTATS (DB2 Utilities)

SYSIBM.SYSTABLEPART table (DB2 SQL)

SYSIBM.SYSTABLES table (DB2 SQL)

SYSIBM.SYSTABSTATS table (DB2 SQL)

Related information:

DSNU234I (DB2 Messages)

DSNU244I (DB2 Messages)

Chapter 26. Designing indexes for performance

Indexes can provide efficient data access in many situations, and DB2 uses them for other purposes. However, certain costs are also associated with creating and maintaining indexes, and you must consider these costs in your database design.

About this task

PSPI

Indexes can provide efficient access to data through index access, data clustering, and ordering retrieved data without a sort operation. DB2 also uses indexes for other purposes, such as to enforce uniqueness the uniqueness of column values, as in the case of parent keys, and to partition tables.

However, before you begin to create indexes, you must carefully consider their costs. You might be able to eliminate indexes that are no longer necessary or change the characteristics of an index to reduce disk usage.

Dropping unneeded indexes also improves performance because of savings in index maintenance.

Procedure

When designing or evaluating indexes:

- Consider the following costs of indexes:
  - Indexes require storage space. Padded indexes require more space than non-padded indexes for long index keys. For short index keys, non-padded indexes can take more space.
  - Each index requires an index space and a data set, or as many data sets as the number of data partitions if the index is partitioned, and operating system restrictions exist on the number of open data sets.
  - Indexes must be changed to reflect every insert or delete operation on the base table. If an update operation updates a column that is in the index, then the index must also be changed. The time required by these operations increases accordingly.
  - Indexes can be built automatically when loading data, but this process takes time. They must be recovered or rebuilt if the underlying table space is recovered, which might also be time-consuming.
- When analyzing index access, ask the following questions:
  - Can you consolidate indexes by including non-key columns on unique indexes?
  - Would adding a column to an index allow the query to use index-only access?
  - Do you need a new index?
  - Is your choice of clustering index correct?

Related concepts:

Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX')

Indexes on table columns (DB2 Administration Guide)

Creation of indexes (Introduction to DB2 for z/OS)

Related tasks:

Altering DB2 indexes (DB2 Administration Guide)

Analyzing index impact (DB2 Query Workload Tuner for z/OS)

Related reference:

DB2 Query Workload Tuner for z/OS

Related information:

Implementing DB2 indexes (DB2 Administration Guide)

Choosing index page sizes

With the CREATE INDEX statement, you can specify buffer pool sizes of 4 KB, 8 KB, 16 KB, and 32 KB for indexes.

About this task

Compressed indexes can use 8 KB, 16 KB, or 32 KB buffer pools. When you use compressed indexes, more index leaf pages can be stored on a disk. The index leaf pages are compressed down from either 8 KB, 16 KB, or 32 KB to fit into a 4 KB page size. For an 8 KB index page size, the best compression ratio is 2:1. For a 16 KB index page size, the best compression ratio is 4:1, and for a 32 KB page size, the best compression ratio is 8:1.

Indexes that are not compressed can take advantage of the larger page sizes of 8 KB, 16 KB, and 32 KB, without the concern of wasted space in the buffer pool. When index compression is working to its fullest potential for any given buffer pool page size, space is not wasted. However, for any combination of buffer pool page sizes (8 KB, 16 KB, and 32 KB) that are compressed to a 4 KB disk page size, some space in the buffer pool or on the disk will be wasted.

For example, consider the following scenarios for a compressed index that has an 8 KB buffer pool page size:

- If the index compression is very marginal, only half of the 8 KB page in the buffer pool will be used, because only 4 KB of space on the disk is used.
- If the index compression is very good (for example, a ratio of 4:1), only half of the 4 KB page on the disk is used.
- If the index compression is almost exactly 2:1, neither space in the buffer pool nor on the disk is wasted. A full 8 KB page of index keys can be compressed to reside on the disk using only a 4 KB page size.

Procedure

To choose appropriate page sizes for indexes, use the following approaches:

- Specify larger page sizes for indexes with sequential insert and fetch patterns. A larger page size can also yield a larger fanout in index non-leaf pages, which can reduce the number of levels in an index and improve performance.

  Creating an index with a larger page size could also reduce the number of page splits in the index. A reduction in page splits is especially beneficial if the latch contention from the index splits is frequent. For example:

  - Latch class 6 in data sharing

  - Latch class X'46' in IFCID 57 performance trace record in a data sharing environment
  - Latch class X'FE' in IFCID 57 record in a non-data-sharing environment

  It can also lead to better performance for sequential access to the index.
- Specify smaller pages sizes for indexes with random fetch patterns.

Example

GUPI

The following example specifies a 16 KB buffer pool for the index being created:

```
CREATE INDEX INDEX1 ON TABLE1 ( I1 ASC, I2 ASC) BUFFERPOOL BP16K1
```

You can specify a 4 KB, 8 KB, 16 KB, or 32 KB default buffer pool for indexes in a particular database using the CREATE DATABASE or ALTER DATABASE by using the INDEXBP option as in the following examples:

```
CREATE DATABASE MYDB INDEXBP BP16K1
ALTER DATABASE MYDB INDEXBP BP16K1
```

GUPI

Related tasks:

Compressing indexes

Assigning database objects to buffer pools

Related reference:

CREATE DATABASE (DB2 SQL)

CREATE INDEX (DB2 SQL)

ALTER INDEX (DB2 SQL)

ALTER DATABASE (DB2 SQL)

Related information:

Implementing DB2 indexes (DB2 Administration Guide)

Reserving free spaces for indexes

You might reduce performance problems that result from index page splits by reserving free space when you reorganize the indexes.

About this task

When sufficient free space is unavailable in an index space, index page splits can result and performance might suffer.

With the introduction of fast storage hardware that enable fast list prefetch, index organization becomes less important. A disorganized index is likely to have plenty of free space. Therefore, you do not need to reorganize indexes for the purpose of creating free space.

However, free space remains an important consideration whenever you do reorganize your indexes.

Procedure

To manage free space for indexes, use the following approaches:

- Reserve some free space in cases when reorganizing an index would result in a subsequent increase in index page splits. Reserving free space can reduce the frequency of index page splits.
- Specify no free space in the following cases:
  - A non-clustering index contains a column with a timestamp value that causes the inserts into the index to be in sequence.
  - Inserts are in ascending order by key of the clustering index or they are caused by LOAD RESUME SHRLEVEL NONE and update activity is only on fixed-length columns with non-compressed data.
  - The associated table is read-only, or you do not plan to insert new data or update the existing data in the table.
  - The data is stored on storage hardware that uses fast controllers and solid state disks and can process list prefetch efficiently.

Related concepts:

LEAFNEAR and LEAFFAR columns

List prefetch (PREFETCH='L' or 'U')

Related tasks:

Reserving free space for table spaces

Maintaining data organization

Determining when to reorganize indexes

Related reference:

ALTER INDEX (DB2 SQL)

CREATE INDEX (DB2 SQL)

REORG INDEX (DB2 Utilities)

SYSIBM.SYSINDEXPART table (DB2 SQL)

Related information:

DB2 for z/OS and List Prefetch Optimizer (IBM Redbooks)

Eliminating unnecessary partitioning indexes

You can reclaim the space used by partitioning indexes by converting to table-controlled partitioning. A partitioning index requires as many data sets as the partitioned table space.

About this task

In *table-controlled partitioning*, partitioning key and limit keys are specified in the CREATE TABLESPACE statement and not the CREATE INDEX statement, which eliminates the need for any partitioning index data sets.

Procedure

To eliminate the disk space that is required for the partitioning index data sets:

1. Drop partitioning indexes that are used solely to partition the data, and not to access it.
2. Convert to table-controlled partitioning.

Related concepts:

Differences between partitioning methods (DB2 Administration Guide)

Automatic conversion to table-controlled partitioning (DB2 Administration Guide)

Related tasks:

Creating tables that use table-controlled partitioning (DB2 Administration Guide)

Related reference:

CREATE TABLESPACE (DB2 SQL)

Indexes to avoid sorts

In addition to selective access to data, indexes can also order data, and sometime eliminate the need to sort the data.

PSPI Some sorts can be avoided if index keys are in the order needed by ORDER BY, GROUP BY, a join operation, or DISTINCT in an aggregate function. In other cases, such as when list prefetch is used, the index does not provide useful ordering, and the selected data might have to be sorted.

When it is absolutely necessary to prevent a sort, consider creating an index on the column or columns necessary to provide that ordering. Consider also using the clause OPTIMIZE FOR 1 ROW to discourage DB2 from choosing a sort for the access path.

Consider the following query:

```
SELECT  C2, SUM(C3)
FROM T1
WHERE C1 = 17
GROUP BY C2;
```

An ascending index on C1 or an index on (C1,C2,C3®) could eliminate a sort.

Backward index scan

In some cases, DB2 can use a backward index scan on a descending index to avoid a sort on ascending data. Similarly, an ascending index can be used to avoid a sort on descending data. For DB2 to use a backward index scan, the following conditions must be true:

- The index includes the columns in the ORDER BY clause in the same order that they appear in the ORDER BY clause.
- Each column in the sequence must have the opposite sequence (ASC or DESC) of the ORDER BY clause.

Example: backward index scan

Suppose that an index exists on the ACCT_STAT table. The index is defined by the following columns: ACCT_NUM, STATUS_DATE, STATUS_TIME. All of the columns in the index are in ascending order. Now, consider the following SELECT statements:

```
SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE DESC, STATUS_TIME DESC;
```

```
SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE ASC, STATUS_TIME ASC;
```

By using a backward index scan, DB2 can use the same index for both statements.

Randomized index key columns

You might also be able to avoid a sort by using the RANDOM option to create an index with a randomized key column, as long as the randomized key column is not included within an ORDER BY clause.

Example: randomized index key columns

You can avoid sorts in query that uses GROUP BY processing by using an index with a randomized key. Consider the following statements:

```
CREATE INDEX I1
ON T1(C1, C2 RANDOM, C3);
```

```
SELECT  C2, SUM(C3)
FROM T1
WHERE C1 = 17
GROUP BY C2;
```

The query can use index I1 because all equal values of the original column C2 are stored contiguously on the index, and have identical random values stored. Although, the order of the query's output would appear to be arbitrary (as opposed to the output if an ASC or DESC index was used), the correctness of the results is not effected. Only the order in which the result tuples are represented to the application is effected by the randomization. If you want to see the results in order, you must enforce the order with an ORDER BY statement, which requires a sort.

When sorts are more efficient

Not all sorts are inefficient. For example, if the index that provides ordering is not an efficient one and many rows qualify, it is possible that using another access path to retrieve and then sort the data could be more efficient than the inefficient, ordering index.

Indexes that are created to avoid sorts can sometimes be non-selective. If these indexes require data access and if the cluster ratio is poor, these indexes are unlikely to be chosen. Accessing many rows by using a poorly clustered index is often less efficient than accessing rows by using a table space scan and sort. Both table space scan and sort benefit from sequential access. PSPI

Related concepts:

Ways to order rows (Introduction to DB2 for z/OS)

Related tasks:

Minimizing the cost of retrieving few rows

Dropping indexes that were created to avoid sorts

Dropping indexes that were created to avoid sorts

Indexes that are defined only to avoid a sort for queries with an ORDER BY clause are unnecessary if DB2 can perform a backward scan of another index to avoid the sort.

About this task

In earlier versions of DB2, you might have created ascending and descending versions of the same index for the sole purpose of avoiding a sort operation.

Procedure

To recover the space that is used by these indexes:

Drop indexes that were created to avoid sorts.
For example, consider the following query:

```
SELECT C1, C2, C3 FROM T
   WHERE C1 > 1
   ORDER BY C1 DESC;
```

Having an ascending index on C1 would not have prevented a sort to order the data. To avoid the sort, you needed a descending index on C1. DB2 can scan an index either forwards or backwards, which can eliminate the need to have indexes with the same columns but with different ascending and descending characteristics.
For DB2 to be able to scan an index backwards, the index must be defined on the same columns as the ORDER BY and the ordering must be exactly opposite of what is requested in the ORDER BY. For example, if an index is defined as `C1 DESC, C2 ASC`,DB2 can use:

- A forward scan of the index for `ORDER BY C1 DESC, C2 ASC`
- A backward scan of the index for `ORDER BY C1 ASC, C2 DESC`

However, DB2 does need to sort for either of the following ORDER BY clauses:

- `ORDER BY C1 ASC, C2 ASC`
- `ORDER BY C1 DESC, C2 DESC`

Related concepts:

Indexes to avoid sorts

Ways to order rows (Introduction to DB2 for z/OS)

Related reference:

order-by-clause (DB2 SQL)

Saving disk space by using non-Padded indexes

You can save disk space by using non-padded indexes instead of padded indexes.

About this task

Introductory concepts

Indexes that are padded or not padded (Introduction to DB2 for z/OS)

When you define an index as NOT PADDED, the varying-length columns in the index are not padded to their maximum length. If the index contains at least one varying-length column, the length information is stored with the key.

Consequently, the amount of savings depends on the number of varying-length columns in the index and the actual length of the columns in those indexes versus their maximum lengths.

Procedure

To use index padding efficiently:

As a general rule, use non-padded indexes only if the average amount that is saved is greater than about 18 bytes per column. For example, assume that you have an index key that is defined on a VARCHAR(128) column and the actual length of the key is 8 bytes. An index that is defined as NOT PADDED would require approximately 9 times less storage than an index that is defined as PADDED, as shown by the following calculation:

```
(128 + 4) / (8 + 2 + 4) = 9
```

Related concepts:

What happens to an index on altered columns (DB2 Administration Guide)

Related tasks:

Altering how varying-length index columns are stored (DB2 Administration Guide)

Compressing indexes

You can compress your indexes to significantly reduce the physical space requirements for most indexes.

Before you begin

Use the DSN1COMP utility on existing indexes to get an indication of the appropriate page size for new indexes. You can choose 8K, 16K, and 32K buffer pool page sizes for the index. Choosing a 32K or 16K buffer pool instead of a 8K buffer pool accommodates a potentially higher compression ratio, but also increases the potential to use more storage. Estimates for index space savings from the DSN1COMP utility, whether on the true index data or some similar index data, are not exact.

About this task

Index compression is heavily data-dependent, and some indexes might contain data that will not yield significant space savings. Compressed indexes might also use more real and virtual storage than non-compressed indexes. The amount of additional real and virtual storage used depends on the compression ratio used for the compressed keys, the amount of free space, and the amount of space used by the key map. The recommendation is to use index compression where a reduction in index storage consumption is more important than a possible decrease in index performance.

The additional cost of compressed indexes can be zero even in random key updates, as long as index pages can be kept in the buffer pool.

Procedure

GUPI To specify index compression:

Specify the compression option by issuing a CREATE INDEX or ALTER INDEX statement.

COMPRESS YES

Activates index compression. The buffer pool used to create the index must be 8 KB, 16 KB, or 32 KB in size. The physical page size on disk will be 4 KB. If you create the index with the clause COMPRESS YES, index compression begins as soon as the first index entries are added.

Restrictions:

- For user-managed index data sets, a compressed index requires a defined control interval size (CISZ) of 4 KB.
- For DB2-managed index data sets that are altered to enable compression (ALTER COMPRESS YES), the next utility operation to remove the REBUILD-pending state will not apply the utility REUSE option.

COMPRESS NO

Specifies that no index compression will be in effect. This is the default option for the CREATE INDEX statement.

GUPI

If you activate or deactivate compression with an ALTER INDEX statement, the index is placed into REBUILD-pending (RBDP) status for partitioned indexes and page set REBUILD-pending (PSRBD) status for non-partitioned indexes. You need to use the REBUILD INDEX utility to rebuild the index, or use the REORG utility to reorganize the table space that corresponds to the index.

Related tasks:

Compressing your data

Choosing index page sizes

Related reference:

ALTER INDEX (DB2 SQL)

CREATE INDEX (DB2 SQL)

DSN1COMP (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

Related information:

Implementing DB2 indexes (DB2 Administration Guide)

Index splitting for sequential INSERT activity

DB2 detects sequential inserts and splits index pages asymmetrically to improve space usage and reduce split processing.

You can further improve performance by choosing the appropriate page size for index pages.

When all the entries in a leaf page are consumed during inserts, DB2 allocates a new page and moves some entries from the old page to the new page. DB2 detects when a series of inserts adds keys in ascending or descending sequential order.

When such a pattern is detected, DB2 splits the index pages asymmetrically, by placing more or fewer keys on the newly allocated page. In this way, DB2 allocates page space more efficiently and reduces the frequency of split processing operations.

Traditionally, DB2 split index pages by moving approximately half the entries to the new page. According to that logic, when sequential inserts added keys in ascending order, the freed space in the old index page was never used. This meant that an index used only half of the allocated page space. Page-splitting also occurred more frequently because the index would fill the available half of each newly allocated page very quickly.

Larger index page sizes can be beneficial in cases where a frequent index split results from heavy inserts. The frequency of index splitting can be determined from LEAFNEAR, LEAFFAR, and NLEAF in SYSINDEXES and SYSINDEXPART catalog tables, latch 70 (and latch class 6 in statistics) contention in data sharing, and latch 254 contention in non data sharing (in latch class 7 in statistics) from performance trace.

A smaller index page size can be beneficial for achieving higher buffer pool hit ratios in random read-intensive applications.

Related concepts:

Relief for sequential key insert (DB2 for z/OS What's New?)

Related tasks:

Choosing index page sizes

Related reference:

SYSIBM.SYSINDEXES table (DB2 SQL)

SYSIBM.SYSINDEXPART table (DB2 SQL)

Creating indexes to improve referential integrity performance for foreign keys

When you define foreign keys, you can improve the performance of certain operations by creating indexes that match the columns of the foreign key.

About this task

For operations that require referential integrity checks, DB2 uses available indexes to improve the performance of the checking operations. For primary keys, the indexes are required for referential integrity, so they are always available for DB2 to use. However, indexes are not required for foreign keys, but you can create them to improve the performance of the checks for operations on the parent table.

Introductory concepts

Creation of relationships with referential constraints (Introduction to DB2 for z/OS)

DB2 keys (Introduction to DB2 for z/OS)

Creation of indexes (Introduction to DB2 for z/OS)

Procedure

For rows of a parent table that are frequently deleted, create indexes that support the performance of referential integrity:

Create an index on the columns of a foreign key. DB2 can use the index to improve the performance of the operations that check the validity of the DELETE statement and its possible effect on the dependent table. Use the following information to help you plan your approach:

- For the index to qualify, the leading columns of the index must be identical to and in the same order as all columns in the foreign key. The index can include more columns, but the leading columns must match the definition of the foreign key.

  Restriction: Indexes that use expressions cannot be used for this purpose.
- A foreign key can also be the primary key. In that case, the primary index is also a unique index on the foreign key, and every row of the parent table has at most one dependent row. The dependent table might be used to hold information that pertains to only a few of the occurrences of the entity that is described by the parent table. For example, a dependent of the employee table might contain information that applies only to employees working in a different country.
- If the first *n* columns of the foreign key are the same as the columns of the primary key, the primary key can share columns of the foreign key.

For example, the following CREATE TABLE statement specifies constraint names REPAPA and REPAE for the foreign keys in the employee-to-project activity table.

```
CREATE TABLE DSN8A10.EMPPROJACT
      (EMPNO     CHAR(6)         NOT NULL,
       PROJNO    CHAR(6)         NOT NULL,
       ACTNO     SMALLINT        NOT NULL,
       CONSTRAINT REPAPA FOREIGN KEY (PROJNO, ACTNO)
               REFERENCES DSN8A10.PROJACT ON DELETE RESTRICT,
       CONSTRAINT REPAE FOREIGN KEY (EMPNO)
               REFERENCES DSN8A10.EMP ON DELETE RESTRICT)
  IN DATABASE DSN8D10A;
```

In the sample project activity table, the primary index (on PROJNO, ACTNO, ACSTDATE) serves as an index on the foreign key on PROJNO. It does not serve as an index on the foreign key on ACTNO, because ACTNO is not the first column of the index.

Related concepts:

Referential constraints (Introduction to DB2 for z/OS)

Related tasks:

Defining a foreign key (DB2 Application programming and SQL)

Creating DB2 indexes (DB2 Administration Guide)

Related reference:

CREATE INDEX (DB2 SQL)

Enabling efficient access for queries on star schemas

Pair-wise join processing simplifies index design by using single-column indexes to join a fact table and the associated dimension tables according to AND predicates.

Procedure

PSPI

To design indexes to enable pair-wise join:

1. Create an index for each key column in the fact able that corresponds to a dimension table.
2. Partition by, and cluster data according to, commonly used dimension keys. Doing so can reduce the I/O that is required on the fact table for pair-wise join.

What to do next

If you require further performance improvement for some star schema queries, consider the following index design recommendations to encourage DB2 to use star join access:

- Define a multi-column index on all key columns of the fact table. Key columns are fact table columns that have corresponding dimension tables.
- If you do not have information about the way that your data is used, first try a multi-column index on the fact table that is based on the correlation of the data. Put less highly correlated columns later in the index key than more highly correlated columns.
- As the correlation of columns in the fact table changes, reevaluate the index to determine if columns in the index should be reordered.
- Define indexes on dimension tables to improve access to those tables.
- When you have executed a number of queries and have more information about the way that the data is used, follow these recommendations:
  - Put more selective columns at the beginning of the multi-column index.
  - If a number of queries do not reference a dimension, put the column that corresponds to that dimension at the end of the index, or remove it completely.

PSPI

Indexes for efficient star schema processing

You can create indexes to enable DB2 to use special join methods for star schemas.

PSPI

A *star schema* is a database design that, in its simplest form, consists of a large table called a *fact table,* and two or more smaller tables, called *dimension tables.* More complex star schemas can be created by breaking one or more of the dimension tables into multiple tables.

To access the data in a star schema design, you often write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables. These types of queries are known as *star-join queries.*

For a star-join query, DB2 might use special join types, *star join* and *pair-wise join*, if the following conditions are true:

- The tables meet the conditions of a star join. (JOIN_TYPE='S')

- The tables meet the conditions of a pair-wise join. (JOIN_TYPE='P')
- The STARJOIN system parameter is set to ENABLE, and the number of tables in the query block is greater than or equal to the minimum number that is specified in the SJTABLES system parameter.

Whether DB2 uses star join, pair-wise join, or traditional join methods for processing a star schema query is based on which method results in the lowest cost access path. The existence of a star schema does not guarantee that either star join or pair-wise join access will be chosen.

PSPI

Related concepts:

Star schema access

Star join access (JOIN_TYPE='S')

Pair-wise join access (JOIN_TYPE='P')

Related reference:

STAR JOIN QUERIES field (STARJOIN subsystem parameter) (DB2 Installation and Migration)

SJTABLES in macro DSN6SPRM (DB2 Installation and Migration)

Part 6. Programming applications for performance

You can achieve better DB2 performance by considering performance as you program and deploy your applications.

Procedure

To improve the performance of application programs that access data in DB2, use the following approaches when writing and preparing your programs:

- Program your applications for concurrency. The goal is to program and prepare applications in a way that:
  - Protects the integrity of the data that is being read or updated from being changed by other applications.
  - Minimizes the length of time that other access to the data is prevented.

  For more information about data concurrency in DB2 and recommendations for improving concurrency in your application programs, see the following topics:
  - Designing databases for concurrency
  - Concurrency and locks
  - Improving concurrency
  - Improving concurrency in data sharing environments (DB2 Data Sharing Planning and Administration)
- Write SQL statements that access data efficiently. The predicates, subqueries, and other structures in SQL statements affect the access paths that DB2 uses to access the data.

  For information about how to write SQL statements that access data efficiently, see the following topics:
  - Ways to improve query performance (Introduction to DB2 for z/OS)
  - Writing efficient SQL queries
- Use EXPLAIN or SQL optimization tools to analyze the access paths that DB2 chooses to process your SQL statements. By analyzing the access path that DB2 uses to access the data for an SQL statement, you can discover potential problems. You can use this information to modify your statement to perform better.

  For information about how you can use EXPLAIN tables, and SQL optimization tools such as IBM Data Studio or IBM Data Server Manager, to analyze the access paths for your SQL statements, see the following topics:
  - Investigating access path problems
  - 00C200A4 (DB2 Codes)
  - Investigating SQL performance by using EXPLAIN
  - Interpreting data access by using EXPLAIN
  - EXPLAIN tables
  - EXPLAIN (DB2 SQL)
  - Tuning SQL with Optim Query Tuner, Part 1: Understanding access paths (IBM developerWorks)
  - Generating visual representations of access plans (IBM Data Studio)
- Consider performance in the design of applications that access distributed data. The goal is to reduce the amount of network traffic that is required to access the

distributed data, and to manage the use of system resources such as distributed database access threads and connections.

  For information about improving the performance of applications that access distributed data, see the following topics:
  - Ways to reduce network traffic (Introduction to DB2 for z/OS)
  - Managing DB2 threads
  - Improving performance for applications that access distributed data
  - Improving performance for SQL statements in distributed applications
- Use stored procedures to improve performance, and consider performance when creating stored procedures.

  For information about stored procedures and DB2 performance, see the following topics:
  - Implementing DB2 stored procedures ()
  - Improving the performance of stored procedures and user-defined functions

Related concepts:

Query and application performance analysis (Introduction to DB2 for z/OS)

Programming for the instrumentation facility interface (IFI)

Related tasks:

Overview of programming applications that access DB2 for z/OS data (DB2 Application programming and SQL)

Setting limits for system resource usage by using the resource limit facility

Planning for and designing DB2 applications (DB2 Application programming and SQL)

Chapter 27. Programming for concurrency

You can design your application programs to protect the integrity of accessed data without preventing other processes from accessing the same data for long periods of time.

About this task

Concurrency is the ability of more than one application process to access the same data at essentially the same time. Concurrency must be controlled to prevent lost updates and such possibly undesirable effects as unrepeatable reads and access to uncommitted data.

One basic way that DB2 controls concurrency is by using locks for units of work. When a unit of work completes, all locks that were implicitly acquired by that unit of work are released, which enables a new unit of work to begin. The amount of processing time that is used by a unit of work in your program affects the length of time that DB2 prevents other users from accessing that locked data. When several programs try to use the same data concurrently, each program's unit of work should be as short as possible to minimize the interference between the programs.

Procedure

PSPI

To design your applications for concurrency:

- Program applications to access data in the same order. When two application access the same rows of a table in the same order, one application might need to wait for the other, but they cannot deadlock. Therefore the recommendation is to try to program different applications to access rows in the same order, and access tables in the same order.
- Commit work as soon as doing so is practical, to avoid unnecessary lock contention, even in read-only applications.

  Taking commit points frequently in a long running unit of recovery (UR) has the following benefits at the possible cost of more CPU usage and log write I/Os:

  – Reduces lock contention, especially in a data sharing environment

  – Improves the effectiveness of lock avoidance, especially in a data sharing environment

  – Reduces the elapsed time for DB2 system restart following a system failure

  – Reduces the elapsed time for a unit of recovery to rollback following an application failure or an explicit rollback request by the application

  – Provides more opportunity for utilities, such as online REORG, to break in

  Consider using the URCHKTH subsystem parameter or the URLGWTH subsystem parameter to identify applications that do not committing frequently. URCHKTH identifies when too many checkpoints have occurred without a UR issuing a commit. It is helpful in monitoring overall system activity. URLGWTH enables detects applications that might write too many log records between commit points, potentially creating a lengthy recovery situation for critical tables.

Even though an application might conform to the commit frequency standards of the installation under normal operational conditions, variation can occur based on system workload fluctuations. For example, a low-priority application might issue a commit frequently on a system that is lightly loaded. However, under a heavy system load, the use of the CPU by the application might be preempted, and, as a result, the application might violate the rule set by the URCHKTH subsystem parameter. For this reason, add logic to your application to commit based on time elapsed since last commit, and not solely based on the amount of SQL processing performed. In addition, take frequent commit points in a long running unit of work that is read-only to reduce lock contention and to provide opportunities for utilities, such as online REORG, to access the data.

Committing frequently is equally important for objects that are not logged and objects that are logged. Make sure, for example, that you commit work frequently even if the work is done on a table space that is defined with the NOT LOGGED option. Even when a given transaction modifies only tables that reside in not logged table spaces, a unit of recovery is still established before updates are performed. Undo processing will continue to read the log in the backward direction looking for undo log records that must be applied until it detects the beginning of this unit of recovery as recorded on the log. Therefore, such transactions should perform frequent commits to limit the distance undo processing might have to go backward on the log to find the beginning of the unit of recovery.

- Include logic in your application program to retry after a deadlock or timeout to attempt recovery from the contention situation without assistance. Such a method could help you recover from the situation without assistance from operations personnel. You can use the following methods to determine whether a timeout or deadlock occurs:
  - The SQLERRD(3) field in the SQLCA
  - A GET DIAGNOSTICS statement
- Bind most applications with the ISOLATION(CS) and CURRENTDATA(NO) options. These options enable DB2 to release locks early and avoid taking locks in many cases. ISOLATION(CS) typically enables DB2 to release acquired locks as soon as possible. The CURRENTDATA(NO) typically enables DB2 to acquire the fewest number of locks, for better *lock avoidance*. When you use ISOLATION(CS) and CURRENTDATA(NO), consider using the SKIPUNCI subsystem parameter value to YES so that readers do not wait for the outcome of uncommitted inserts.
- If you do not use ISOLATION(CS) and CURRENTDATA(NO), use the following bind options, in order of decreasing preference:
  1. ISOLATION(CS) with CURRENTDATA(YES), when data returned to the application must not be changed before your next FETCH operation.
  2. ISOLATION(RS), when data returned to the application must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.
  3. ISOLATION(RR), when data evaluated as the result of a query must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.
- Use ISOLATION(UR) option cautiously. The Resource Recovery Services attachment facility UR isolation acquires almost no locks on rows or pages. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and end users can accept the logical inconsistencies that can occur.

As an alternative, consider using a SKIP LOCKED DATA clause if omitting data is preferable to reading uncommitted data in your application.

- Use sequence objects to generate unique, sequential numbers. Using an identity column is one way to generate unique sequential numbers.

  However, as a column of a table, an identity column is associated with and tied to the table, and a table can have only one identity column. Your applications might need to use one sequence of unique numbers for many tables or several sequences for each table. As a user-defined object, sequences provide a way for applications to have DB2 generate unique numeric key values and to coordinate the keys across multiple rows and tables.

  The use of sequences can avoid the lock contention problems that can result when applications implement their own sequences, such as in a one-row table that contains a sequence number that each transaction must increment. With DB2 sequences, many users can access and increment the sequence concurrently without waiting. DB2 does not wait for a transaction that has incremented a sequence to commit before allowing another transaction to increment the sequence again.
- Examine multi-row operations such as multi-row inserts, positioned updates, and positioned deletes, which have the potential of expanding the unit of work. This situation can affect the concurrency of other users that access the data. You can minimize contention by adjusting the size of the host-variable-array, committing between inserts, updates, and preventing lock escalation.
- Use global transactions, which enables DB2 and other transaction managers to participate in a single transaction and thereby share the same locks and access the same data. The Resource Recovery Services attachment facility (RRSAF) relies on a z/OS component called Resource Recovery Services (RRS). RRS provides system-wide services for coordinating two-phase commit operations across z/OS products. For RRSAF applications and IMS transactions that run under RRS, you can group together a number of DB2 agents into a single global transaction.

  A global transaction allows multiple DB2 agents to participate in a single global transaction and thus share the same locks and access the same data. When two agents that are in a global transaction access the same DB2 object within a unit of work, those agents do not deadlock or timeout with each other. The following restrictions apply:
  - Parallel Sysplex® is not supported for global transactions.
  - Because each of the "branches" of a global transaction are sharing locks, uncommitted updates issued by one branch of the transaction are visible to other branches of the transaction.
  - Claim/drain processing is not supported across the branches of a global transaction, which means that attempts to issue CREATE, DROP, ALTER, GRANT, or REVOKE might deadlock or timeout if they are requested from different branches of the same global transaction.
  - LOCK TABLE might deadlock or timeout across the branches of a global transaction.
- Use optimistic concurrency control. *Optimistic concurrency control* represents a faster, more scalable locking alternative to database locking for concurrent data access. It minimizes the time for which a given resource is unavailable for use by other transactions.

  When an application uses optimistic concurrency control, locks are obtained immediately before a read operation and released immediately. Update locks are obtained immediately before an update operation and held until the end of the transaction. Optimistic concurrency control uses the RID and a row change token to test whether data has been changed by another transaction since the last read operation.

Because DB2 can determine when a row was changed, you can ensure data integrity while limiting the time that locks are held. With optimistic concurrency control, DB2 releases the row or page locks immediately after a read operation. DB2 also releases the row lock after each FETCH, taking a new lock on a row only for a positioned update or delete to ensure data integrity.

To implement optimistic concurrency control, you must establish a row change timestamp column with a CREATE TABLE statement or an ALTER TABLE statement. The column must be defined with one of the following null characteristics:

– NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

– NOT NULL GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

After you establish a row change timestamp column, DB2 maintains the contents of this column. When you want to use this change token as a condition when making an update, you can specify an appropriate predicate for this column in a WHERE clause.

Related concepts:

Lock modes

SQL communication area (SQLCA) (DB2 SQL)

Related tasks:

Improving concurrency

Choosing an ISOLATION option

Choosing a RELEASE option

Related reference:

UR LOG WRITE CHECK field (URLGWTH subsystem parameter) (DB2 Installation and Migration)

UR CHECK FREQ field (URCHKTH subsystem parameter) (DB2 Installation and Migration)

COMMIT (DB2 SQL)

GET DIAGNOSTICS (DB2 SQL)

LOCK TABLE (DB2 SQL)

PREPARE (DB2 SQL)

Bind options for locks

Certain BIND options determine when an application process acquires and releases its locks and how it isolates its actions from effects of concurrent processes.

Choosing a RELEASE option

The RELEASE bind option controls when an application releases locks that it acquires on objects such as partitions, tables, or table spaces that it accesses. It applies only to partition, table, and table space locks.

About this task

PSPI

An application program acquires table, partition, or table space locks only when it accesses the specific objects. In most cases, the least restrictive lock mode that is required to process each SQL statement is used. However, a statement sometimes reuses a more restrictive lock than required, if a suitable lock remains from a previous statement that accessed the same objects.

The RELEASE option applies only to static SQL statements, which are bound before your program runs. Dynamic SQL statements acquire locks only when they access objects and release the locks at the next commit point. However, locks for dynamic statements might be held past commit. When the KEEPDYNAMIC(YES) bind option is specified and the value of the CACHEDYN subsystem parameter is YES, the RELEASE(DEALLOCATE) bind option is honored for dynamic SELECT, INSERT, UPDATE, and DELETE statements.

The RELEASE option does not apply to page, row, LOB, or XML locks.

Procedure

Choose a value for the RELEASE bind option that is appropriate for the characteristics of the particular application:

RELEASE(DEALLOCATE)
This option results in the most efficient use of processing time in most cases. The locks are released only when the application ends and the object is deallocated.

RELEASE(COMMIT)
This option provides the greatest concurrency. However, if the application commits frequently, RELEASE(COMMIT) requires more processing time. Under this option, locks are released at different times, depending on the attachment facility:

TSO, Batch, and CAF

An SQL COMMIT or ROLLBACK statement is issued, or your application process ends.

IMS A CHKP or SYNC call (for single-mode transactions), a GU call to the I/O PCB, or a ROLL or ROLB call is completed

CICS A SYNCPOINT command is issued.

Cursors that are defined under the WITH HOLD option are an exception. Locks that are necessary to maintain the cursor position are held past the commit point.

Example

Consider an application that selects employee names and telephone numbers from a table. Assume that employees can update their own telephone numbers, and they can run several searches in succession. The application is bound with the RELEASE(DEALLOCATE) bind option because most uses of this application do not update and do not commit. For those uses, little difference exists between RELEASE(COMMIT) and RELEASE(DEALLOCATE).

However, administrators might update several phone numbers in one session with the application, and the application commits after each update. In that case, RELEASE(COMMIT) releases a lock that DB2 must acquire again immediately. RELEASE(DEALLOCATE) holds the lock until the application ends, avoiding the processing that is required to release and acquire the lock several times.

PSPI

Related concepts:

The duration of a lock

Related reference:

RELEASE bind option (DB2 Commands)

KEEPDYNAMIC bind option (DB2 Commands)

CACHE DYNAMIC SQL field (CACHEDYN subsystem parameter) (DB2 Installation and Migration)

Choosing an ISOLATION option

Various *isolation levels* offer less or more concurrency at the cost of more or less protection from other application processes.

About this task

PSPI

The *ISOLATION option* of an application specifies the degree to which operations are isolated from the possible effects of other operations that act concurrently. The ISOLATION options specified how soon DB2 can release S and U locks on rows or pages. Regardless of the isolation level that you specify, outstanding claims on DB2 objects can inhibit the execution of DB2 utilities or commands.

The default ISOLATION option differs for different types of bind operations, as shown in the following table.

Table 57. The default ISOLATION values for different types of bind operations

Operation	**Default value**
BIND PLAN	ISOLATION (CS) with CURRENTDATA (NO)
BIND PACKAGE	The value used by the plan that includes the package in its package list
REBIND PLAN or PACKAGE	The existing value for the plan or package being rebound

The recommended order of preference for isolation levels is:

1. Cursor stability (CS)
2. Uncommitted read (UR)
3. Read stability (RS)
4. Repeatable read (RR)

Although uncommitted read provides the lowest level of isolation, cursor stability isolation is recommended in most cases. ISOLATION(CS) provides a high level of concurrency, without sacrificing data integrity.

Procedure

To ensure that your applications can access your data concurrently, without sacrificing data integrity:

1. Choose an isolation level according to the needs and characteristics of the particular application.
2. Bind most applications with the ISOLATION(CS) and CURRENTDATA(NO) options. These options enable DB2 to release locks early and avoid taking locks in many cases. ISOLATION(CS) typically enables DB2 to release acquired locks as soon as possible. The CURRENTDATA(NO) typically enables DB2 to acquire the fewest number of locks, for better *lock avoidance*. When you use ISOLATION(CS) and CURRENTDATA(NO), consider using the SKIPUNCI subsystem parameter value to YES so that readers do not wait for the outcome of uncommitted inserts.
3. If you do not use ISOLATION(CS) and CURRENTDATA(NO), use the following bind options, in order of decreasing preference:
   a. ISOLATION(CS) with CURRENTDATA(YES), when data returned to the application must not be changed before your next FETCH operation.
   b. ISOLATION(RS), when data returned to the application must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.
   c. ISOLATION(RR), when data evaluated as the result of a query must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.
4. Use ISOLATION(UR) option cautiously. The Resource Recovery Services attachment facility UR isolation acquires almost no locks on rows or pages. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and end users can accept the logical inconsistencies that can occur.

   As an alternative, consider using a SKIP LOCKED DATA clause if omitting data is preferable to reading uncommitted data in your application.

Related tasks:

Choosing a CURRENTDATA option

Improving concurrency for applications that tolerate incomplete results

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

BIND PACKAGE (DSN) (DB2 Commands)

BIND PLAN (DSN) (DB2 Commands)

REBIND PACKAGE (DSN) (DB2 Commands)

REBIND PLAN (DSN) (DB2 Commands)

SKIP LOCKED DATA (DB2 SQL)

The ISOLATION (CS) option

The ISOLATION (CS) or *cursor stability* option allows maximum concurrency with data integrity. Under the ISOLATION (CS) option, a transaction holds locks only on its uncommitted changes and on the current row of each of its cursors.

GUPI

However, after the process leaves a row or page, another process can change the data. With CURRENTDATA(NO), the process does not have to leave a row or page to allow another process to change the data. If the first process returns to read the same row or page, the data is not necessarily the same. Consider the following consequences of that possibility:

- For table spaces created with LOCKSIZE ROW, PAGE, or ANY, a change might occur even while executing a single SQL statement that reads the same row multiple times. In the following statement, data read by the inner SELECT might be changed by another transaction before it is read by the outer SELECT.

  ```
  SELECT * FROM T1
    WHERE C1 = (SELECT MAX(C1) FROM T1);
  ```

  Therefore, the information returned by this query might be from a row that is no longer the one with the maximum value for C1.

- Another case is a process that reads a row and returns later to update it. That row might no longer exist or might not exist in the state that it did when your application process originally read the row. That is, another application might have deleted or updated the row. If your application is doing non-cursor operations on a row under the cursor, make sure that the application can tolerate "not found" conditions.

  Similarly, assume that another application updates a row after it is read by your application. If your application returns later to update the row based on the read-in value, your application erases the update from the second application.

  Therefore, if you use ISOLATION(CS) with update, your process might need to lock out concurrent updates. One method is to declare a cursor with the FOR UPDATE clause.

For packages and plans that contain updatable static scrollable cursors, ISOLATION(CS) enables DB2 to use *optimistic concurrency control*. DB2 can use optimistic concurrency control to shorten the amount of time that locks are held in the following situations:

- Between consecutive fetch operations
- Between fetch operations and subsequent positioned update or delete operations

DB2 cannot use optimistic concurrency control for dynamic scrollable cursors. With dynamic scrollable cursors, the most recently fetched row or page from the base table remains locked to maintain position for a positioned update or delete.

The two following figures show processing of positioned update and delete operations without optimistic concurrency control and with optimistic concurrency control.

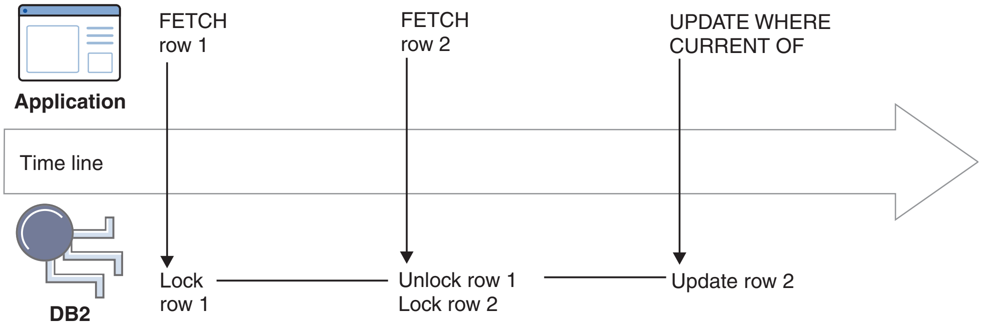


Figure 15. Positioned updates and deletes with a static non-scrollable cursor and without optimistic concurrency control

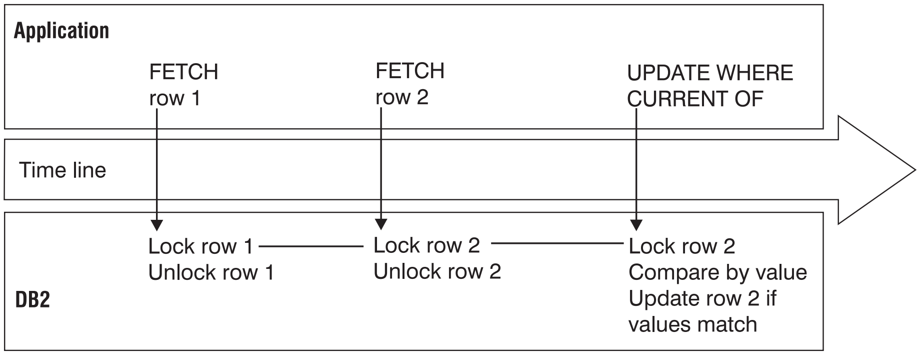


Figure 16. Positioned updates and deletes with a static sensitive scrollable cursor and with optimistic concurrency control

Optimistic concurrency control consists of the following steps:

1. When the application requests a fetch operation to position the cursor on a row, DB2 locks that row, executes the fetch operation, and releases the lock.
2. When the application requests a positioned update or delete operation on the row, DB2 performs the following steps:
   a. Locks the row.
   b. Re-evaluates the predicate to ensure that the row still qualifies for the result table.

GUPI

Related concepts:

Lock size

Enhancements to optimistic concurrency control and update detection (DB2 for z/OS What's New?)

Related tasks:

Choosing a CURRENTDATA option

Specifying the size of locks for a table space

Enabling block fetch for distributed applications

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

The ISOLATION (UR) option

The ISOLATION (UR) or *uncommitted read* option allows an application to read while acquiring few locks, at the risk of reading uncommitted data. UR isolation applies only to the following read-only operations: SELECT, SELECT INTO, or FETCH from a read-only result table.

Reading uncommitted data introduces an element of uncertainty.

For example, an application tracks the movement of work from station to station along an assembly line. As items move from one station to another, the application subtracts from the count of items at the first station and adds to the count of items at the second station. Assume that you want to query the count of items at all the stations, while the application is running concurrently.

If your query reads data that the application has changed but has not committed:

- If the application subtracts an amount from one record before adding it to another, the query could miss the amount entirely.
- If the application adds first and then subtracts, the query could add the amount twice.
- If the application updates the record in a way that causes the record to move to another page, the record could be temporarily invisible to the query.

If those situations can occur and are unacceptable, do not use UR isolation.

Restrictions for using ISOLATION (UR)

You cannot use the ISOLATION (UR) option for the following types of statements:

- INSERT, UPDATE, DELETE, and MERGE
- SELECT FROM INSERT, UPDATE, DELETE, or MERGE.
- Any cursor defined with a FOR UPDATE clause

If you bind with ISOLATION(UR) and the statement does not specify WITH RR or WITH RS, DB2 uses CS isolation for these types of statements.

When an application uses uncommitted read isolation and runs concurrently with applications that update variable-length records such that the update creates a double-overflow record, the ISOLATION(UR) application might miss rows that are being updated.

When to use ISOLATION (UR)

You can probably use UR isolation in cases such as the following examples:

When errors cannot occur

The follow examples describe situations in which errors can be avoided while using the ISOLATION(UR) option.

Reference tables

Like a table of descriptions of parts by part number. Such tables are rarely updated, and reading an uncommitted update is probably no more damaging than reading the table 5 seconds earlier.

Tables with limited access

The employee table of Spiffy Computer, our hypothetical user. For security reasons, updates can be made to the table only by members of a single department. And that department is also the only one that can query the entire table. It is easy to restrict queries to times when no updates are being made and then run with UR isolation.

When an error is acceptable

Spiffy Computer wants to do some statistical analysis on employee data. A typical question is, "What is the average salary by sex within education level?" Because reading an occasional uncommitted record cannot affect the averages much, UR isolation can be used.

When the data already contains inconsistent information

Spiffy computer gets sales leads from various sources. The data is often inconsistent or wrong, and users of the data are accustomed to dealing with that problem. Inconsistent access to a table of data on sales leads does not add to the problem.

When not to use ISOLATION (UR)

Do not use uncommitted read, ISOLATION (UR), in the following cases:

- When computations must balance
- When the answer must be accurate
- When you are unsure whether using the ISOLATION (UR) might cause damage

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

The ISOLATION (RS) option

The ISOLATION (RS) or *read stability* option enables an application to read the same pages or rows more than once and prevents updates or deletes to qualifying rows by other processes. However, other applications can insert or update rows that did not satisfy the search condition of the original application.

Read stability isolation might result in greater concurrency than repeatable read. Other applications cannot change rows that are returned to the original application. However, they can insert new rows or update rows that did not satisfy the original search condition. Only rows or pages that satisfy the stage 1 predicate (and all rows or pages evaluated during stage 2 processing) are locked until the application commits. The following figure illustrates this process. In the example, the rows held by locks L2 and L4 satisfy the predicate.

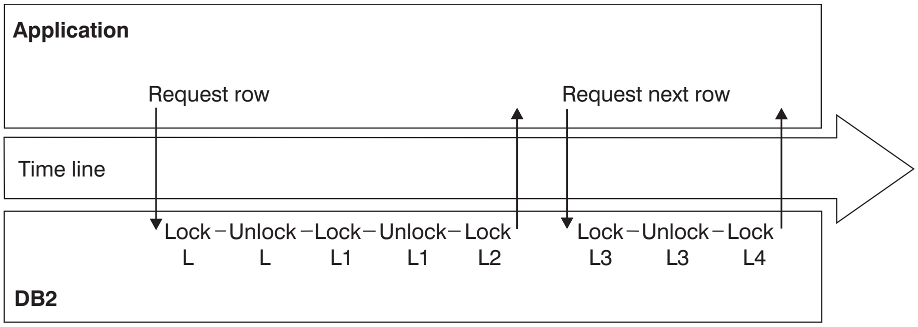


Figure 17. How an application that uses RS isolation acquires locks when no lock avoidance techniques are used. Locks L2 and L4 are held until the application commits. The other locks aren't held.

Applications that use read stability isolation can leave rows or pages locked for long periods, especially in a distributed environment.

If you do use read stability, plan for frequent commit points.

An installation option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with read stability.

Related concepts:

Stage 1 and stage 2 predicates

Lock avoidance

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

The ISOLATION (RR) option

The ISOLATION (RR) or *repeatable read* option allows the application to read the same pages or rows more than once without allowing any update, insert, or delete operations by other processes. All accessed rows or pages are locked, even if they do not satisfy the predicate. Under the ISOLATION (RR) option, the data that an application references cannot be updated by any other applications before the application reaches a commit point.

GUPI

Applications that use repeatable read might leave rows or pages locked for longer periods, especially in a distributed environment. They also might claim more logical partitions than similar applications that use cursor stability isolation.

Applications that use repeatable read and access a nonpartitioned index cannot run concurrently with utility operations that drain all claim classes of the nonpartitioned index. This restriction remains true even if the application and utility are accessing different logical partitions. For example, an application bound with ISOLATION(RR) cannot update partition 1 while the LOAD utility loads data into partition 2. Concurrency is restricted because the utility needs to drain all the repeatable-read applications from the nonpartitioned index to protect the repeatability of the reads by the application.

They are also subject to being drained more often by utility operations.

Because so many locks can be taken, lock escalation might take place. Frequent commits release the locks and can help avoid lock escalation.

With repeatable read, lock promotion occurs for table space scan to prevent the insertion of rows that might qualify for the predicate. (If access is via index, DB2 locks the key range. If access is via table space scans, DB2 locks the table, partition, or table space.)

An installation option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with repeatable read.

Repeatable read and CP parallelism

For CP parallelism, locks are obtained independently by each task. This situation can possibly increase the total number of locks taken for applications that have the following attributes:

- Use the repeatable read isolation level.
- Use CP parallelism.
- Repeatedly access the table space and use a lock mode of IS, without issuing COMMIT statements.

Repeatable read or read stability isolation cannot be used with Sysplex query parallelism.

GUPI

Related concepts:

Parallel processing

Utility operations with nonpartitioned indexes

Related tasks:

Enabling parallel processing

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

Phenomena that might occur with isolation levels other than repeatable read

Because isolation levels other than repeatable read allow updates by other applications while an application is reading data, the application that is reading data might retrieve more or fewer rows than are expected.

The phenomena that can occur are called *phantom rows*, *dirty read*, and *non-repeatable read*.

Phantom rows

SQL transaction T1 reads a set of rows that satisfy some search condition on a table. SQL transaction T2 then executes SQL statements that generate one or more new rows in the table that also satisfy the search condition that is used by SQL transaction T1. If T1 then repeats the original read with the same search condition, T1 receives a different set of rows.

This phenomenon can occur with uncommitted read (UR), cursor stability (CS), or read stability (RS) isolation.

One case in which phantom rows occur is when there is a multiple-column index on a table, and the following activities occur concurrently:

- One transaction executes a query that selects rows based on the first indexed column of a table. The query uses an index scan.
- Another transaction updates a value in the second indexed column of the table. The corresponding row has not yet been selected by the query. The update causes the index key value for the updated row to move to a point that the index scan has already passed.

In this situation, the query misses the updated row.

Example: Suppose that table EMP_INFO is defined as follows:

```
CREATE TABLE EMP_INFO (
 WORKDEPT CHAR(3) NOT NULL,
 LASTNAME VARCHAR(15),
 FIRSTNME VARCHAR(12),
 JOB CHAR(8));
```

Index EMP_INFO_IX is defined on the first two columns of table EMP_INFO:

```
CREATE INDEX EMP_INFO_IX ON EMP_INFO(WORKDEPT, LASTNAME);
```

Table EMP_INFO contains rows like these. The rows are displayed in index order. This is the order in which the rows are read with an index scan.

WORKDEPT	LASTNAME	FIRSTNME	JOB
A00	HAAS	CHRISTINE	PRES
A00	HEMMINGER	DIAN	SALESREP
A00	LUCCHESI	VINCENZO	SALESREP
A00	O'CONNELL	SEAN	CLERK
A00	ORLANDO	GREG	CLERK
B01	THOMPSON	MICHAEL	MANAGER
C01	KWAN	SALLY	MANAGER
C01	NATZ	KIM	ANALYST
C01	NICHOLLS	HEATHER	ANALYST
C01	QUINTANA	DOLORES	ANALYST
...			

The packages that execute transactions T1 and T2 are bound with RS isolation.

Transaction T1 retrieves a list of employees who are in department A00 by executing the following query:

```
SELECT FIRSTNME, LASTNAME FROM EMP_INFO WHERE WORKDEPT = 'A00';
```

The query uses index EMP_INFO_IX to retrieve the rows.

The following actions occur:

1. The select operation reads the index first key value, ('A00','HAAS'), and retrieves the first row:

FIRSTNME	LASTNAME
CHRISTINE	HAAS

2. At the same time, transaction T2 updates the last name of employee O'Connell to Connelly:

   ```
   UPDATE EMP_INFO SET LASTNAME='CONNELLY' WHERE LASTNAME='O''CONNELL';
   ```

   The update operation also uses index EMP_INFO_IX.
3. The update operation changes the order of the keys in index EMP_INFO_IX to this order:

```
('A00','CONNELLY')
('A00','HAAS')
('A00','HEMMINGER')
('A00','LUCCHESI')
('A00','ORLANDO')
...
```

4. The select operation continues to retrieve rows using the updated index order:

FIRSTNME	LASTNAME
DIAN	HEMMINGER
VINCENZO	LUCCHESI
GREG	ORLANDO

   The row for Sean Connelly is a phantom row. The result set does not contain that row because the index scan has already passed the new position of the row's index key value.

5. If the query is executed again, with no updates that affect the order of the index key values, all rows are returned:

FIRSTNME	LASTNAME
SEAN	CONNELLY
CHRISTINE	HAAS
DIAN	HEMMINGER
VINCENZO	LUCCHESI
GREG	ORLANDO

Dirty read

SQL transaction T1 modifies a row. SQL transaction T2 reads that row before T1 executes a commit operation. If T1 then executes a rollback operation, T2 will have read a row that was never committed, and therefore can be considered never to have existed.

This phenomenon can occur with uncommitted read (UR) isolation.

Example: Suppose that table EMP_INFO is defined as in the previous example, and contain the same data.

The packages that execute transactions T1 and T2 are bound with UR isolation.

The following actions occur:

1. Transaction T1 updates the last name of employee O'Connell to Connelly:

   ```
   UPDATE EMP_INFO SET LASTNAME='CONNELLY' WHERE LASTNAME='O''CONNELL';
   ```

2. Transaction T2 executes the following query:

   ```
   SELECT FIRSTNME, LASTNAME FROM EMP_INFO WHERE WORKDEPT = 'A00';
   ```

   The following rows are returned:

FIRSTNME	LASTNAME
SEAN	CONNELLY
CHRISTINE	HAAS
DIAN	HEMMINGER

FIRSTNME	LASTNAME
VINCENZO	LUCCHESI
GREG	ORLANDO

3. Transaction T1 executes a rollback operation, which reverts this update statement:

   ```
   UPDATE EMP_INFO SET LASTNAME='CONNELLY' WHERE LASTNAME='O''CONNELL';
   ```

   The result set in the previous step is no longer valid, because there is no longer a row with a LASTNAME value of 'CONNELLY'.

Non-repeatable read

SQL transaction T1 reads a row. SQL transaction T2 then modifies or deletes that row and executes a commit operation. If T1 then attempts to reread that row, T1 might receive the modified value or discover that the row has been deleted.

This phenomenon can occur with uncommitted read (UR) or cursor stability (CS) isolation.

Example: Suppose that table EMP_INFO is defined as in the first example, and contain the same data.

The packages that execute transactions T1 and T2 are bound with CS isolation.

The following actions occur:

1. Transaction T1 executes the following query:

   ```
   SELECT FIRSTNME, LASTNAME, JOB FROM EMP_INFO WHERE LASTNAME = 'HAAS';
   ```

   The following row is returned:

FIRSTNME	LASTNAME	JOB
CHRISTINE	HAAS	PRES

2. Transaction T2 updates the Christine Haas' job from PRES to CEO, and commits the update:

   ```
   UPDATE EMP_INFO SET JOB='CEO' WHERE LASTNAME='HAAS' AND FIRSTNME='CHRISTINE';
   COMMIT;
   ```

3. Transaction T1 executes the following query again:

   ```
   SELECT FIRSTNME, LASTNAME, JOB FROM EMP_INFO WHERE LASTNAME = 'HAAS';
   ```

   The following row is returned, which is different from the row that was previously returned by the same query:

FIRSTNME	LASTNAME	JOB
CHRISTINE	HAAS	CEO

Related concepts:

The ISOLATION (RR) option

The ISOLATION (CS) option

The ISOLATION (UR) option

The ISOLATION (RS) option

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

Choosing a CURRENTDATA option

The *CURRENTDATA option* of an application specifies whether data currency is required for read-only and ambiguous cursors when the ISOLATION(CS) option is used. This option enables a trade-off between the improved ability of multiple applications to access the same data concurrently and the risk that non-current data might be returned to the application.

About this task

PSPI

Generally, the CURRENTDATA(NO) option increases the ability of multiple applications to access the same data concurrently. However, the trade off is an increased risk that non-current data might be returned to the application. The CURRENTDATA(YES) reduces the risk of non-current data being returned to the application. However, the trade off is a reduced ability for multiple applications to access the same data concurrently.

The CURRENTDATA bind option applies differently for applications that access local and remote data. For requests to remote systems, the CURRENTDATA has an effect for ambiguous cursors that use the following ISOLATION options: RR, RS, or CS. For access to a remote table or index, CURRENTDATA(YES) turns off block fetching for ambiguous cursors. The data returned with the cursor is current with the contents of the remote table or index for ambiguous cursors. Turning on block fetch offers best performance, but it means the cursor is not current with the base table at the remote site.

PSPI

Procedure

For improved concurrent data access, use the following approaches:

- Bind most applications with the ISOLATION(CS) and CURRENTDATA(NO) options. These options enable DB2 to release locks early and avoid taking locks in many cases. ISOLATION(CS) typically enables DB2 to release acquired locks as soon as possible. The CURRENTDATA(NO) typically enables DB2 to acquire the fewest number of locks, for better *lock avoidance*. When you use ISOLATION(CS) and CURRENTDATA(NO), consider using the SKIPUNCI subsystem parameter value to YES so that readers do not wait for the outcome of uncommitted inserts.
- Commit work as soon as doing so is practical, to avoid unnecessary lock contention, even in read-only applications.

Related concepts:

Lock avoidance

The ISOLATION (CS) option

Problems with ambiguous cursors

Related tasks:

Choosing an ISOLATION option

Related reference:

CURRENTDATA bind option (DB2 Commands)

ISOLATION bind option (DB2 Commands)

The CURRENTDATA option for local access

For local access, the CURRENTDATA option specifies whether data under a cursor must remain *current* with the data in the local base table.

For cursors positioned on data in a work file, the CURRENTDATA option has no effect. This effect applies only to read-only or ambiguous cursors in plans or packages bound with the ISOLATION(CS) option.

CURRENTDATA (YES)

CURRENTDATA (YES) means that the data upon which the cursor is positioned cannot change while the cursor is positioned on it. If the cursor is positioned on data in a local base table or index, then the data returned with the cursor is current with the contents of that table or index. If the cursor is positioned on data in a work file, the data that is returned by cursor matches only the current contents of the work file. The data is not necessarily current with the contents of the underlying table or index.

The following figure shows locking with CURRENTDATA(YES).

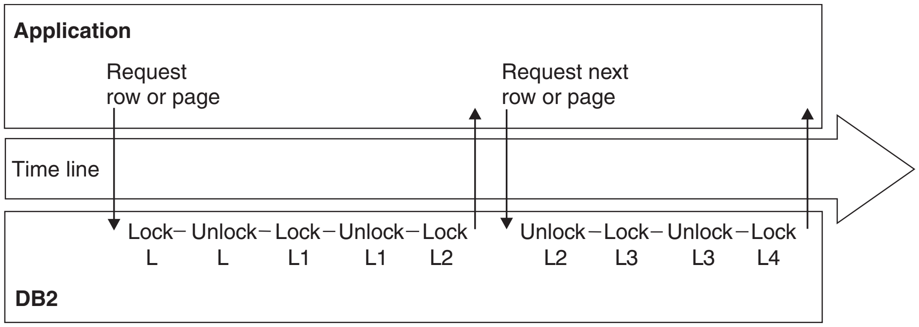


Figure 18. How an application that uses CS isolation with CURRENTDATA (YES) acquires locks. The figure shows access to the base table. The L2 and L4 locks are released after DB2 moves to the next row or page. When the application commits, the last lock is released.

As with work files, if a cursor uses query parallelism, data might not be current with the contents of the table or index, regardless of whether a work file is used. Therefore, for work file access or for parallelism on read-only queries, the CURRENTDATA option has no effect.

If you are using parallelism but want to maintain currency with the data, you have the following options:

- Disable parallelism by one of the following methods:
  - Issue the following SQL statement: `SET CURRENT DEGREE = '1'`
  - Bind the application with the DEGREE(1) bind option.
- Use isolation RR or RS (parallelism can still be used).
- Use a LOCK TABLE statement (parallelism can still be used).

CURRENTDATA(NO)

This option means that the data at the cursor position can change while the cursor is positioned on it.

Related concepts:

Problems with ambiguous cursors

Lock avoidance

Related tasks:

Choosing an ISOLATION option

Related reference:

CURRENTDATA bind option (DB2 Commands)

SET CURRENT DEGREE (DB2 SQL)

CURRENT DEGREE field (CDSSRDEF subsystem parameter) (DB2 Installation and Migration)

CURRENTDATA for remote access

For requests to remote systems, the CURRENTDATA option applies to ambiguous cursors that use the following ISOLATION options: RR, RS, or CS.

GUPI

For access to a remote table or index, CURRENTDATA(YES) turns off block fetching for ambiguous cursors. The data returned with the cursor is current with the contents of the remote table or index for ambiguous cursors. Turning on block fetch offers best performance, but it means that the cursor is not current with the base table at the remote site.

GUPI

Related concepts:

Problems with ambiguous cursors

Block fetch (Introduction to DB2 for z/OS)

Related tasks:

Choosing an ISOLATION option

Enabling block fetch for distributed applications

Related reference:

CURRENTDATA bind option (DB2 Commands)

Lock avoidance

Concurrency is improved when applications are created so that DB2 can use *lock avoidance* techniques.

GUPI

The CURRENTDATA(NO) option enables greater opportunity for avoiding locks. DB2 can check whether a row or page contains committed data. DB2 does not need to obtain any lock on the data when the row or page contains committed data. Unlocked data is returned to the application, and the data can be changed while the cursor is positioned on the row.

To take the best advantage of this method of avoiding locks, make sure all applications that access data concurrently issue COMMIT statements frequently.

The following figure shows how DB2 can avoid taking locks and the following table summarizes the factors that influence lock avoidance.

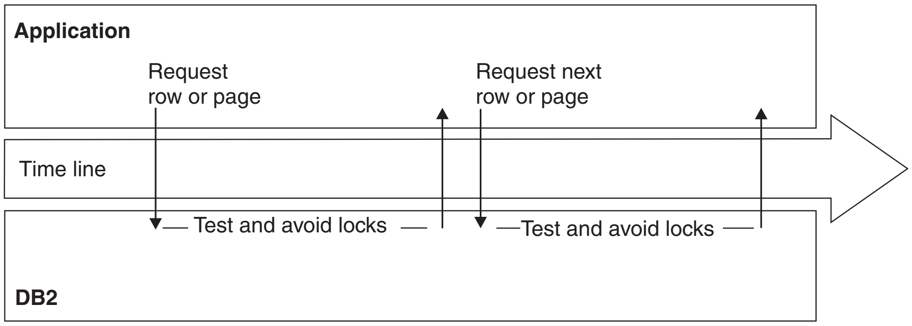


Figure 19. Best case of avoiding locks by using the ISOLATION(CS) and CURRENTDATA(NO) options. This figure shows access to the base table. If DB2 must take a lock, locks are released when DB2 moves to the next row or page, or when the application commits. This behavior matches the CURRENTDATA(YES) option.

Table 58. Lock avoidance factors. "Returned data" means data that satisfies the predicate. "Rejected data" is that which does not satisfy the predicate.

Isolation	CURRENTDATA	Cursor type	Avoid locks on returned data?	Avoid locks on rejected data?
UR	N/A	Read-only	N/A	N/A
CS	YES	Any	No	Yes[1]
	NO	Read-only	Yes	
		Updatable	No	
		Ambiguous	Yes	
RS	N/A	Any	No	Yes[1, 2]
RR	N/A	Any	No	No

Notes:

1. Locks are avoided when the row is disqualified after stage 1 processing
2. Under the ISOLATION(RS) option and multi-row fetch, DB2 releases locks on Stage 1 qualified rows that later fail to qualify for stage 2 predicates at the next fetch of the cursor.

GUPI

Related concepts:

Stage 1 and stage 2 predicates

Related tasks:

Choosing an ISOLATION option

Related reference:

isolation-clause (DB2 SQL)

BIND and REBIND options for packages and plans (DB2 Commands)

Problems with ambiguous cursors

A cursor is considered *ambiguous* if DB2 cannot tell whether it is used for update or read-only purposes.

If the cursor appears to be used only for read-only, but dynamic SQL could modify data through the cursor, then the cursor is ambiguous. If you use CURRENTDATA to indicate an ambiguous cursor is read-only when it is actually targeted by dynamic SQL for modification, you'll get an error. Ambiguous cursors can sometimes prevent DB2 from using lock avoidance techniques. However, misuse of an ambiguous cursor can cause your program to receive a -510 SQLCODE, meaning:

- The plan or package is bound with CURRENTDATA(NO)
- An OPEN CURSOR statement is performed before a dynamic DELETE WHERE CURRENT OF statement against that cursor is prepared
- One of the following conditions is true for the open cursor:
  - Lock avoidance is successfully used on that statement.
  - Query parallelism is used.
  - The cursor is distributed, and block fetching is used.

In all cases, it is a good programming technique to eliminate the ambiguity by declaring the cursor with either the FOR FETCH ONLY or the FOR UPDATE clause.

Related concepts:

Lock avoidance

Related tasks:

Enabling block fetch for distributed applications

Choosing a CURRENTDATA option

Related reference:

OPEN (DB2 SQL)

read-only-clause (DB2 SQL)

Conflicting plan and package bind options

A plan bound with one set of options can include packages in its package list that were bound with different sets of options.

PSPI

In general, statements in a DBRM bound as a package use the options that the package was bound with.

For example, the plan value for CURRENTDATA has no effect on the packages executing under that plan. If you do not specify a CURRENTDATA option explicitly when you bind a package, the default is CURRENTDATA(NO).

The rules are slightly different for the bind options RELEASE and ISOLATION. The values of those two options are set when the lock on the resource is acquired and usually stay in effect until the lock is released. But a conflict can occur if a statement that is bound with one pair of values requests a lock on a resource that is already locked by a statement that is bound with a different pair of values. DB2 resolves the conflict by resetting each option with the available value that causes the lock to be held for the greatest duration.

The table below shows how conflicts between isolation levels are resolved. The first column is the existing isolation level, and the remaining columns show what happens when another isolation level is requested by a new application process.

Table 59. Resolving isolation conflicts

	UR	**CS**	**RS**	**RR**
UR	n/a	CS	RS	RR
CS	CS	n/a	RS	RR
RS	RS	RS	n/a	RR
RR	RR	RR	RR	n/a

PSPI

Using SQL statements to override isolation levels

You can override the isolation level with which a plan or package is bound.

Procedure

PSPI

To override the isolation level for a specific SQL statement:

- Issue the SQL statements, and include a WITH *isolation level* clause. The WITH *isolation level* clause:
  - Can be used on these statements:
    - SELECT
    - SELECT INTO
    - Searched DELETE
    - INSERT from fullselect
    - Searched UPDATE
  - Cannot be used on subqueries.
  - Can specify the isolation levels that specifically apply to its statement. (For example, because WITH UR applies only to read-only operations, you cannot use it on an INSERT statement.)
  - Overrides the isolation level for the plan or package only for the statement in which it appears.

  The following statement finds the maximum, minimum, and average bonus in the sample employee table.

  ```
  SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
    INTO :MAX, :MIN, :AVG
    FROM DSN8A10.EMP
       WITH UR;
  ```

  The statement is executed with uncommitted read isolation, regardless of the value of ISOLATION with which the plan or package containing the statement is bound.
- If you use the WITH RR or WITH RS clause, you can issue SELECT and SELECT INTO statements, and specify the following options:
  - USE AND KEEP EXCLUSIVE LOCKS
  - USE AND KEEP UPDATE LOCKS
  - USE AND KEEP SHARE LOCKS

To use these options, specify them as shown in the following example:

```
SELECT ...
 WITH RS USE AND KEEP UPDATE LOCKS;
```

Results

By using one of these options, you tell DB2 to acquire and hold a specific mode of lock on all the qualified pages or rows. The following table shows which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause.

Table 60. Which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause

Option Value	Lock Mode
USE AND KEEP EXCLUSIVE LOCKS	X
USE AND KEEP UPDATE LOCKS	U
USE AND KEEP SHARE LOCKS	S

With read stability (RS) isolation, a row or page that is rejected during stage 2 processing might still have a lock held on it, even though it is not returned to the application.

With repeatable read (RR) isolation, DB2 acquires locks on all pages or rows that fall within the range of the selection expression.

All locks are held until the application commits. Although this option can reduce concurrency, it can prevent some types of deadlocks and can better serialize access to data.

PSPI

Related tasks:

Choosing an ISOLATION option

Related reference:

ISOLATION bind option (DB2 Commands)

isolation-clause (DB2 SQL)

Controlling concurrent access to tables

You can use LOCK TABLE statements and ISOLATION bind options to prevent other application processes from changing or reading rows in a table or partition while your application is accessing it.

About this task

GUPI

You might want to lock a table or partition so that a single application thread has exclusive access to the contents of an entire table throughout a unit of work, and all concurrent changes are prevented.

Procedure

To control concurrent access to tables, use any of the following approaches:

- Issue LOCK TABLE statements for the table. A new lock is requested immediately when the LOCK TABLE statement is issued, unless a suitable lock already exists. The RELEASE bind option determines when locks that are acquired for LOCK TABLE statements are released.

  Share mode is recommended when your application needs to prevent changes to the entire table but other processes can be allowed to read the data.

  Applications that use the CURRENTDATA(NO) bind option might still be able to read the data, even when exclusive mode locks are used.

  When the goal is to prevent timeouts caused by contention with other applications, either share or exclusive mode can be used. However, exclusive mode is recommended when your application updates the data.

  The locks that are acquired when you issue a LOCK TABLE statement depend on the mode that is requested and the type of table space. The following table shows the modes of locks acquired in segmented and nonsegmented table spaces for LOCK TABLE statements.

Table 61. Modes of locks acquired by LOCK TABLE. LOCK TABLE on partitions behave the same as nonsegmented table spaces.

LOCK TABLE In	Nonsegmented or Universal Table Spaces	Segmented Table Space Tables	Segmented Table Space Table Spaces
EXCLUSIVE MODE	X	X	IX
SHARE MODE	S or SIX[1]	S or SIX[1]	IS

  Note:

  1. The SIX lock is acquired if the process already holds an IX lock. SHARE MODE has no effect if the process already has a lock of mode SIX, U, or X.

  For segmented table spaces that are not partitioned, the LOCK TABLE statement applies to individual tables. For all other table space types, the LOCK TABLE statement applies to the table space or to the specified partition.

- Use application BIND options to prevent access to the table. You can use the following approaches:
  - Bind the application with the ISOLATION(RR) bind option. For table space scans, this option acquires gross locks on the accessed tables, and might impact the concurrency of the application.
  - Design the application to use separate packages and access the table from only a few of the packages. Then bind only those packages with the ISOLATION(RR) or ISOLATION(RS) bind options, and bind the plan with the ISOLATION(CS) option.
- Use LOCKSIZE TABLESPACE for tables that require read-only access. This approach is best for tables that contain relatively static data that is updated only infrequently and only by a process that requires exclusive control over the table.

GUPI

Related concepts:

Lock escalation

Related tasks:

Choosing an ISOLATION option

Explicitly locking LOB tables

Explicitly locking XML data

Related reference:

ISOLATION bind option (DB2 Commands)

RELEASE bind option (DB2 Commands)

CURRENTDATA bind option (DB2 Commands)

LOCK TABLE (DB2 SQL)

isolation-clause (DB2 SQL)

Explicitly locking LOB tables

The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables.

About this task

PSPI

You might use the LOCK table statement for lobs for any of the following reasons:

Procedure

To manage locks for auxiliary tables spaces for LOB data, use the following approaches:

- Use LOCK TABLE to control the number of locks acquired on the auxiliary table. By doing so, you can eliminate the need for lower-level LOB locks.
- Use LOCK TABLE IN SHARE MODE to prevent other applications from inserting LOBs. With auxiliary tables, LOCK TABLE IN SHARE MODE does not prevent any changes to the auxiliary table. The statement does prevent LOBs from being inserted into the auxiliary table, but it does not prevent deletes. Updates are generally restricted also, except where the LOB is updated to a null value or a zero-length string.
- Use LOCK TABLE IN EXCLUSIVE MODE to prevent other applications from accessing LOBs. With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.

PSPI

Related concepts:

Locks for LOB data

Large objects (LOBs) (DB2 SQL)

Related tasks:

Controlling the number of LOB locks (DB2 for z/OS What's New?)

Controlling lock size for LOB table spaces

Controlling concurrent access to tables

Related reference:

LOCK TABLE (DB2 SQL)

Explicitly locking XML data

You can use a LOCK TABLE statement to explicitly lock XML data

About this task

Introductory concepts

XML data type (Introduction to DB2 for z/OS)

XML table spaces (Introduction to DB2 for z/OS)

Procedure

PSPI

Use any of the following approaches to manage XML tables spaces by using LOCK TABLE statements:

- Use LOCK TABLE statements to control the number of locks acquired on the auxiliary table. Locking the auxiliary table eliminates the need for lower-level XML locks.
- Issue the following statement to prevent other applications from modifying the data.

  ```
  LOCK TABLE IN SHARE MODE
  ```

- Issue the following statement to prevent other applications from reading the data.

  ```
  LOCK TABLE IN EXCLUSIVE MODE
  ```

  This statement also prevents uncommitted readers from accessing the data in the XML table space. However, it does not prevent readers from accessing versioned XML documents.

  PSPI

Related concepts:

Locks for XML data

XML versions (DB2 Programming for XML)

Related tasks:

Controlling the number of XML locks

Specifying the size of locks for XML data

Controlling concurrent access to tables

Related reference:

LOCK TABLE (DB2 SQL)

Accessing currently committed data to avoid lock contention

You can reduce lock contention that results from uncommitted insert and delete operations by enabling transactions that read data to access the currently committed data rather than waiting for the uncommitted changes to be resolved.

About this task

PSPI

You can control whether a transaction that reads data must wait for locks that are held on that data for uncommitted insert and delete operations. If the transaction does not wait for insert or delete operations to commit and resolve the lock contention, the read operation can complete more quickly, and concurrency improves.

Transactions that read data can avoid waiting for the operations to commit by reading the currently committed data. *Currently committed data* means the data as it was last committed, before the uncommitted change that holds the lock on the row. For uncommitted insert operations, a transaction that reads currently committed data does not read newly inserted rows that are uncommitted.For uncommitted delete operations, a transaction reads the uncommitted deleted rows, as if the delete operation did not happen. Transactions must always wait for uncommitted update operations.

Transactions that read currently committed data must be able to accept data that might be out of date. For applications, procedures, and functions that can tolerate only the most current data, do not enable read transactions to access the currently committed data.

Access to currently committed data is only available on universal table spaces, and applies only to row-level and page-level locks.

PSPI

Procedure

To control how access to uncommitted data is resolved, use the following approaches:

- For improved concurrency, specify that most transactions can read currently committed data.
- Specify that transactions wait for committed inserts in the following situations:
  - If your application or stored procedure can tolerate only the most current data.
  - When one transaction creates another. If the initial transaction passes information to the second transaction by inserting data into a table that the second transaction reads, then the second transaction must wait for the uncommitted inserts.
- Control how transactions from bound application programs react to uncommitted insert and delete operations by specifying the value of the CONCURRENTACCESSRESOLUTION bind option. When you specify USECURRENTLYCOMMITTED for this bind option, the application program can access currently committed data rows that are locked by uncommitted insert or delete operations without waiting for lock contention to resolve. When you specify WAITFOROUTCOME, the transaction waits for the locks that are held by uncommitted operations to be released. You can specify the CONCURRENTACCESSRESOLUTION option on the following commands:
  - BIND PACKAGE
  - BIND PLAN

  - REBIND PACKAGE
  - REBIND PLAN
  - REBIND TRIGGER PACKAGE
- Control how stored procedures and functions react to uncommitted data by specifying the CONCURRENT ACCESS RESOLUTION option. When you specify USE CURRENTLY COMMITTED, the procedure or function can access currently committed data rows that are locked by uncommitted insert or delete operations without waiting for lock contention to resolve. When you specify WAIT FOR OUTCOME, the transaction waits for the locks that are held by all uncommitted operations to be released. You can specify the CONCURRENT ACCESS RESOLUTION option on the following SQL statements
  - CREATE PROCEDURE
  - ALTER PROCEDURE
  - CREATE FUNCTION
  - ALTER FUNCTION
- Control how prepared SQL statements react to uncommitted insert and delete operations by specifying USE CURRENTLY COMMITTED in the *attribute-string* of a PREPARE statement. This option enables the prepared SQL statement to skip data from rows that are locked by uncommitted insert operations and obtain rows that are locked by uncommitted delete operations, instead of waiting for lock contention to resolve. When you specify WAIT FOR OUTCOME, the transaction waits for the locks that are held by all uncommitted operations to be released.
- Control whether applications skip rows that are locked for uncommitted inserts by specifying the value of the SKIPUNCI subsystem parameter. The value of the SKIPUNCI subsystem parameter applies at the subsystem level, and it applies only to insert operations. It applies only when row-level locking is used. However, it is not limited to universal tables spaces. When the value is YES, applications ignore uncommitted inserts as if the insert did not happen. When the value is NO, the application must wait for the insert operations to commit and the locks contention to be resolved. Specify NO if data is frequently modified by delete and insert operations, such that a new image of the data is inserted without the use of update operations. Otherwise, the data might be missed entirely when the uncommitted inserts are skipped.

Results

When different currently committed data options are specified at different levels, the most specific option applies to the transaction. The option that is specified at the statement level applies before the option specified at the package level. The package-level options apply before the value of the SKIPUNCI subsystem parameter value.

The isolation level of a transaction also controls whether it can read currently committed data. Transactions can skip uncommitted insert operations under the ISOLATION(RS) or ISOLATION(CS) options. However, only transactions that use the ISOLATION(CS) and CURRENTDATA(NO) options can read currently committed data when they encounter uncommitted delete operations.

Related tasks:

Choosing an ISOLATION option

Choosing a CURRENTDATA option

Improving concurrency for applications that tolerate incomplete results

Related reference:

CONCURRENTACCESSRESOLUTION bind option (DB2 Commands)

SKIP UNCOMM INSERTS field (SKIPUNCI subsystem parameter) (DB2 Installation and Migration)

CREATE FUNCTION (DB2 SQL)

ALTER FUNCTION (compiled SQL scalar) (DB2 SQL)

CREATE PROCEDURE (SQL - native) (DB2 SQL)

ALTER PROCEDURE (SQL - native) (DB2 SQL)

PREPARE (DB2 SQL)

Improving concurrency for applications that tolerate incomplete results

You can use the SKIP LOCKED DATA option to skip rows that are locked to increase the concurrency of applications and transactions that can tolerate incomplete results.

Before you begin

PSPI

Your application must use one of the following isolation levels:

- Cursor stability (CS)
- Read stability (RS)

The SKIP LOCKED DATA clause is ignored for applications that use uncommitted read (UR) or repeatable read (RR) isolation levels.

About this task

The SKIP LOCKED DATA option allows a transaction to skip rows that are incompatibly locked by other transactions when those locks would hinder the progress of the transaction. Because the SKIP LOCKED DATA option skips these rows, the performance of some applications can be improved by eliminating lock wait time. However, you must use the SKIP LOCKED DATA option only for applications that can reasonably tolerate the absence of the skipped rows in the returned data. If your transaction uses the SKIP LOCKED DATA option, it does not read or modify data that is held by locks.

However, keep in mind that your application cannot rely on DB2 to skip all data for which locks are held. DB2 skips only locked data that would block the progress of the transaction that uses the SKIP LOCKED DATA option. If DB2 determines through lock avoidance that the locked data is already committed, the locked data is not skipped. Instead, the data is returned with no wait for the locks.

Important: When DB2 skips data because of the SKIP LOCKED DATA option, it does not issue a warning. Even if only a subset of the data that satisfies a query is returned or modified, the transaction completes as if no data was skipped. Use the SKIP LOCKED data option only when the requirements and expectations of the application match this behavior.

Procedure

To improve concurrency for applications that require fast results and can tolerate incomplete results:

Specify the SKIP LOCKED DATA clause in one of the following SQL statements:

- SELECT
- SELECT INTO
- PREPARE
- Searched-UPDATE
- Searched-DELETE

You can also use the SKIP LOCKED DATA option with the UNLOAD utility. Lock mode compatibility for transactions that use the SKIP LOCKED DATA option is the same as lock mode compatibility for other page- and row-level locks, except that a transaction that uses the SKIP LOCKED DATA option does not wait for the locks to be released and skips the locked data instead.

Example

Suppose that a table WORKQUEUE exists in a table space with row-level locking and has as part of its definition ELEMENT, PRIORITY and STATUS columns, which contain the following values:

ELEMENT	PRIORITY	STATUS
1	1	OPEN
2	1	OPEN
3	3	OPEN
4	1	IN-ANALYSIS

Suppose that a transaction has issued an UPDATE against ELEMENT 1 to change its STATUS from OPEN to IN-ANALYSIS, and that the UPDATE has not yet committed:

```
UPDATE WORKQUEUE
  SET STATUS = 'IN-ANALYSIS'
  WHERE ELEMENT = 1;
```

Suppose that a second transaction issues the following SELECT statement to find the highest priority work item:

```
SELECT ELEMENT FROM WORKQUEUE
  WHERE PRIORITY = '1' AND STATUS='OPEN'
  SKIP LOCKED DATA;
```

This query locates the row that contains the ELEMENT=2 value, without waiting for the transaction that holds a lock on the row that contains the ELEMENT=1 value to commit or rollback its operation.

However, you cannot always expect DB2 to skip this data. For example, DB2 might use lock avoidance or other techniques to avoid acquiring certain locks.

PSPI

Related tasks:

Choosing an ISOLATION option

Related reference:

SKIP LOCKED DATA (DB2 SQL)

- select-statement (DB2 SQL)
- SELECT INTO (DB2 SQL)
- UPDATE (DB2 SQL)
- DELETE (DB2 SQL)
- PREPARE (DB2 SQL)

Chapter 28. Writing efficient SQL queries

You might be able to rewrite certain SQL statements to enable more efficient access path selection.

Before you begin

| Before rewriting existing SQL statements is response to performance problems: Check the organization of the data and the availability and accuracy of the data statistics.

Related concepts:

Ways to improve query performance (Introduction to DB2 for z/OS)

Investigating SQL performance by using EXPLAIN

Related tasks:

Maintaining data organization and statistics

Generating visual representations of access plans (IBM Data Studio)

Monitoring SQL performance with IBM optimization tools

Modifying catalog statistics to influence access path selection

Related reference:

REORG TABLESPACE (DB2 Utilities)

RUNSTATS (DB2 Utilities)

Related information:

Tuning single SQL statements (IBM Data Studio)

Coding SQL statements to avoid unnecessary processing

By keeping your SQL statements simple, you can limit the amount of processing that they require.

Procedure

To get the best performance from SQL statements:

- Select only columns that are included in the result set or used to calculate the result set.
- Use GROUP BY and ORDER BY clauses only when the grouping and order of the returned data are important.

  Important: When an ORDER BY clause is not specified, DB2 might return data in any order.
- | Specify the DISTINCT option only when the result set must not contain duplicate rows.

Related concepts:

Implications of using SELECT * (DB2 Application programming and SQL)

How a SELECT statement works (Introduction to DB2 for z/OS)

Ways to order rows (Introduction to DB2 for z/OS)

Ways to summarize group values (Introduction to DB2 for z/OS)

Sorts of data

Related tasks:

Eliminating redundant duplicate rows in the result table (DB2 Application programming and SQL)

Related reference:

group-by-clause (DB2 SQL)

order-by-clause (DB2 SQL)

select-clause (DB2 SQL)

Coding queries with aggregate functions efficiently

If your query involves aggregate functions, you can take measure to increase the chances that they are evaluated when the data is retrieved, rather than afterward. Doing that can improve the performance of the query.

About this task

PSPI

In general, a aggregate function performs best when evaluated during data access and next best when evaluated during DB2 sort. Least preferable is to have a aggregate function evaluated after the data has been retrieved. You can use EXPLAIN to determine when DB2 evaluates the aggregate functions.

Queries that involve the functions MAX or MIN might be able to take advantage of one-fetch access.

Procedure

To ensure that an aggregate function is evaluated when DB2 retrieves the data:

Code the query so that every aggregate function that it contains meets the following criteria:

- No sort is needed for GROUP BY. Check this in the EXPLAIN output.
- No stage 2 (residual) predicates exist. Check this in your application.
- No distinct set functions exist, such as COUNT(DISTINCT C1).
- If the query is a join, all set functions must be on the last table joined. Check this by looking at the EXPLAIN output.
- All aggregate functions must be on single columns with no arithmetic expressions.
- The aggregate function is not one of the following aggregate functions:
  - STDDEV
  - STDDEV_SAMP
  - VAR
  - VAR_SAMP

PSPI

Using non-column expressions efficiently

DB2 can evaluate certain predicates at an earlier stage of processing called stage 1, so that the query that contains the predicate takes less time to run. When a predicate contains column and non-column expressions on the same side of the operator, DB2 must evaluate the predicate at a later stage.

Procedure

PSPI

To enable stage 1 processing of queries that contain non-column expressions:

Write each predicate so that all non-column expressions appear on the opposite side of the operator from any column expressions.

Example

The following predicate combines a column, SALARY, with values that are not from columns on one side of the operator:

```
WHERE SALARY + (:hv1 * SALARY) > 50000
```

If you rewrite the predicate in the following way, DB2 can evaluate it more efficiently:

```
WHERE SALARY > 50000/(1 + :hv1)
```

In the second form, the column is by itself on one side of the operator, and all the other values are on the other side of the operator. The expression on the right is called a *non-column expression*.

PSPI

Using predicates efficiently

You can improve how DB2 processes SQL statements by following certain practices when writing predicates.

Procedure

PSPI

To use predicates most efficiently in SQL statements:

- Use stage 1 predicates whenever possible. Stage 1 predicates are better than stage 2 predicates because they disqualify rows earlier and reduce the amount of processing that is needed at stage 2. In terms of resource usage, the earlier a predicate is evaluated, the better.
- Write queries to evaluate the most restrictive predicates first. When predicates with a high filter factor are processed first, unnecessary rows are screened as early as possible, which can reduce processing cost at a later stage. However, a predicate's restrictiveness is only effective among predicates of the same type and at the same evaluation stage.

  PSPI

Predicates and access path selection

Predicates are found in the WHERE, HAVING, or ON clauses of SQL statements; they describe attributes of data.

PSPI

The Predicates of a SQL statement affect how DB2 selects the access path for the statement. Because you can use SQL to express the same query in different ways, knowing how predicates affect path selection helps you write queries that access data efficiently.

Most predicates are based on the columns of a table. They either qualify rows (through an index) or reject rows (returned by a scan) when the table is accessed. The resulting qualified or rejected rows are independent of the access path that is chosen for that table.

The following query has three predicates: an equal predicate on C1, a BETWEEN predicate on C2, and a LIKE predicate on C3.

```
SELECT * FROM T1
  WHERE C1 = 10 AND
        C2 BETWEEN 10 AND 20 AND
        C3 NOT LIKE 'A%'
```

Predicates in a HAVING clause are not used when DB2 selects access paths. The term *predicate* herein refers only to predicates in WHERE or ON clauses. The following attributes of predicates influence access path selection:

- The *type* of predicate, according to its operator or syntax.
- Whether the predicate is *indexable*.
- Whether the predicate is *stage 1* or *stage 2*.
- Whether the predicate contains a rowid column.
- Whether the predicates in part of an ON clause.

The following terms are used to differentiate and classify certain kinds of predicates:

Simple or compound

A *compound* predicate is the result of two predicates, whether simple or compound, that are connected together by AND or OR Boolean operators. All others are *simple*.

Local or join

Local predicates reference only one table. They are local to the table and restrict the number of rows that are returned for that table. *Join predicates* involve more than one table or correlated reference. They determine the way rows are joined from two or more tables.

Boolean term

Any predicate that is not contained by a compound OR predicate structure is a *Boolean term*. If a Boolean term is evaluated false for a particular row, the whole WHERE clause is evaluated false for that row.

Predicates in the ON clause

The ON clause supplies the join condition in an outer join. For a full outer join, the clause can use only equal predicates. For other outer joins, the clause can use any predicates except predicates that contain subqueries.

For inner joins, ON clause predicates can supply the join condition and local filtering, and they are semantically equivalent to WHERE clause predicates.

For full outer join, the ON clause is evaluated during the join operation like a stage 2 predicate.

In an outer join, predicates that are evaluated after the join are stage 2 predicates. Predicates in a table expression can be evaluated before the join and can therefore be stage 1 predicates.

For example, in the following statement, the predicate `EDLEVEL > 100` is evaluated before the full join and is a stage 1 predicate:

```
SELECT * FROM (SELECT * FROM DSN8A10.EMP
   WHERE EDLEVEL  > 100) AS X FULL JOIN DSN8A10.DEPT
      ON X.WORKDEPT  = DSN8A10.DEPT.DEPTNO;
```

PSPI

Related concepts:

Ways to filter the number of returned rows (Introduction to DB2 for z/OS)

Predicates (DB2 SQL)

Interpreting data access by using EXPLAIN

Ways to join data from more than one table (Introduction to DB2 for z/OS)

Related tasks:

Generating visual representations of access plans (IBM Data Studio)

Related reference:

where-clause (DB2 SQL)

having-clause (DB2 SQL)

joined-table (DB2 SQL)

Predicate types

The type of a predicate depends on its operator or syntax. The type determines what type of processing and filtering occurs when DB2 evaluates the predicate.

PSPI

Predicates can be organized into the following types:

Subquery predicates

Any predicate that includes another SELECT statement. For example:

```
C1 IN (SELECT C10 FROM TABLE1)
```

Equal predicates

Any predicate that is not a subquery predicate and has an equal operator and no NOT operator. Also included are predicates of the form C1 IS NULL and C IS NOT DISTINCT FROM. For example:

```
C1=100
```

Assume that a unique index, I1 (C1), exists on table T1 (C1, C2), and that all values of C1 are positive integers. DB2 chooses index access for the following query that contains an equal predicate because the index is highly selective on column C1:

```
SELECT * FROM T1 WHERE C1 = 0;
```

Range predicates

Any predicate that is not a subquery predicate and contains one of the following operators:

>
>=
<
<=
LIKE
BETWEEN

For example:

```
C1>100
```

Assume that a unique index, I1 (C1), exists on table T1 (C1, C2), and that all values of C1 are positive integers. The range predicate in the following query does not eliminate any rows of T1. Therefore, DB2 might determine during access path selection that a table space scan is more efficient than the index scan.

```
SELECT C1, C2 FROM T1 WHERE C1 >= 0;
```

IN predicates

A predicate of the form column IN (list of values). For example:

```
C1 IN (5,10,15)
```

NOT predicates

Any predicate that is not a subquery predicate and contains a NOT operator. Also included are predicates of the form C1 IS DISTINCT FROM. For example:

```
C1 <> 5 or C1 NOT BETWEEN 10 AND 20
```

PSPI

Related reference:

Summary of predicate processing

Indexable and non-indexable predicates

An *indexable* predicate can match index entries; predicates that cannot match index entries are said to be *non-indexable*.

PSPI

To make your queries as efficient as possible, you can use indexable predicates in your queries and create suitable indexes on your tables. Indexable predicates allow the possible use of a matching index scan, which is often a very efficient access path.

Indexable predicates might or might not become matching predicates of an index; depending on the availability of indexes and the access path that DB2 chooses at bind time.

For example, if the employee table has an index on the column LASTNAME, the following predicate can be a matching predicate:

```
SELECT * FROM DSN8A10.EMP WHERE LASTNAME = 'SMITH';
```

In contrast, the following predicate cannot be a matching predicate, because it is not indexable.

```
SELECT * FROM DSN8A10.EMP WHERE SEX <> 'F';
```

PSPI

Related concepts:

Stage 1 and stage 2 predicates

Related reference:

Summary of predicate processing

Stage 1 and stage 2 predicates

Rows retrieved for a query go through two stages of processing. Certain predicates can be applied during the first stage of processing, whereas other cannot be applied until the second stage of processing. You can improve the performance of your queries by using predicates that can be applied during the first stage whenever possible.

PSPI

Predicates that can be applied during the first stage of processing are called *Stage 1 predicates*. These predicates are also sometimes said to be *sargable*. Similarly, predicates that cannot be applied until the second stage of processing are called *stage 2 predicates*, and sometimes described as *nonsargable* or *residual* predicates.

Whether a predicate is stage 1 or stage 2 depends on the following factors:

- The syntax of the predicate.
- Data type and length of constants or columns in the predicate.

  A simple predicate whose syntax classifies it as indexable and stage 1 might not be indexable or stage 1 because of data types that are associated with the predicate. For example, a predicate that is associated with either columns or constants of the DECFLOAT data type is never treated as stage 1. Similarly a predicate that contains constants or columns whose lengths are too long also might not be stage 1 or indexable.

  For example, the following predicate is not indexable, where CHARCOL is defined as CHAR(6):

  ```
  CHARCOL > 'ABCDEFG'
  ```

  For example, The following predicate is not stage 1, If DECCOL is defined as DECIMAL(18,2), because the precision of the decimal column is greater than 15.:

  ```
  DECCOL > 34.5e0,
  ```

- Whether DB2 evaluates the predicate before or after a join operation. A predicate that is evaluated after a join operation is always a stage 2 predicate.
- Join sequence.

  The same predicate might be stage 1 or stage 2, depending on the join sequence. *Join sequence* is the order in which DB2 joins tables when it evaluates a query. The join sequence is not necessarily the same as the order in which the tables appear in the predicate.

For example, the predicate might be stage 1 or stage 2:

```
T1.C1=T2.C1+1
```

If T2 is the first table in the join sequence, the predicate is stage 1, but if T1 is the first table in the join sequence, the predicate is stage 2.

You can determine the join sequence by executing EXPLAIN on the query and examining the resulting plan table.

All indexable predicates are stage 1. The predicate C1 LIKE %BC is stage 1, but is not indexable.

PSPI

Related concepts:

Indexable and non-indexable predicates

Related reference:

Summary of predicate processing

Boolean term predicates

You can improve the performance of queries by choosing Boolean term predicates over non-Boolean term predicates for join operations whenever possible.

PSPI

A *Boolean term predicate* is a simple or compound predicate that, when it is evaluated false for a particular row, makes the entire WHERE clause false for that particular row.

For example, in the following query P1, P2 and P3 are simple predicates:

```
SELECT * FROM T1 WHERE P1 AND (P2 OR P3);
```

- P1 is a simple Boolean term predicate.
- P2 and P3 are simple non-Boolean term predicates.
- P2 OR P3 is a compound Boolean term predicate.
- P1 AND (P2 OR P3) is a compound Boolean term predicate.

Single index processing generally requires Boolean term predicates for matching index access. DB2 rewrites simple non-Boolean term OR conditions against a single column to use Boolean term IN-lists. For example, the following statement is rewritten:

```
SELECT * FROM T1 WHERE C1 = ? OR C1 = ?;
```

The following statement is the result:

```
SELECT * FROM T1 WHERE C1 IN (?, ?)
```

More complex Boolean term predicates might be candidates for multi-index access or range list access.

In join operations, Boolean term predicates can reject rows at an earlier stage than can non-Boolean term predicates.

Recommendation: For join operations, choose Boolean term predicates over non-Boolean term predicates whenever possible.

PSPI

Related concepts:

Predicates that qualify for direct row access

How DB2 modifies IN predicates

Related reference:

Summary of predicate processing

Examples of predicate properties

The included examples can help you to understand how and at which stage DB2 processes different predicates.

PSPI

Assume that predicate P1 and P2 are simple, stage 1, indexable predicates:

P1 AND P2 is a compound, stage 1, indexable predicate.

P1 OR P2 is a compound, stage 1 predicate, not indexable except by a union of RID lists from two indexes.

The following examples of predicates illustrate the general rules of predicate processing. In each case, assume that an index has been created on columns (C1, C2, C3, and C4) of the table and that 0 is the lowest value in each column.

`WHERE C1=5 AND C2=7`

Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.

`WHERE C1=5 AND C2>7`

Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.

`WHERE C1>5 AND C2=7`

Both predicates are stage 1, but only the first matches the index. A matching index scan could be used with C1 as a matching column.

`WHERE C1=5 OR C2=7`

Both predicates are stage 1 but not Boolean terms. The compound is indexable. Multiple-index access for the compound predicate is not possible because no index has C2 as the leading column. For single-index access, C1 and C2 can be only index screening columns.

`WHERE C1=5 OR C2<>7`

The first predicate is indexable and stage 1, and the second predicate is stage 1 but not indexable. The compound predicate is stage 1 and not indexable.

`WHERE C1>5 OR C2=7`

Both predicates are stage 1 but not Boolean terms. The compound is indexable. Multiple-index access for the compound predicate is not possible because no index has C2 as the leading column. For single-index access, C1 and C2 can be only index screening columns.

`WHERE C1 IN (`*`cor subq`*`) AND C2=C1`

As written, both predicates are stage 2 and not indexable. The index is not considered for matching-index access, and both predicates are evaluated at stage 2. However, DB2 might transform the correlated subquery to a non-correlated subquery during processing, in which case both predicates become indexable and stage 1

`WHERE C1=5 AND C2=7 AND (C3 + 5) IN (7,8)`

The first two predicates only are stage 1 and indexable. The index is considered for matching-index access, and all rows satisfying those two predicates are passed to stage 2 to evaluate the third predicate.

`WHERE C1=5 OR C2=7 OR (C3 + 5) IN (7,8)`

The third predicate is stage 2. The compound predicate is stage 2 and all three predicates are evaluated at stage 2. The simple predicates are not Boolean terms and the compound predicate is not indexable.

`WHERE C1=5 OR (C2=7 AND C3=C4)`

The third predicate is stage 2. The two compound predicates `(C2=7 AND C3=C4)` and `(C1=5 OR (C2=7 AND C3=C4))` are stage 2. All predicates are evaluated at stage 2.

`WHERE (C1>5 OR C2=7) AND C3 = C4`

The compound predicate `(C1>5 OR C2=7)` is indexable and stage 1. The simple predicate `C3=C4` is not stage1; so the index is not considered for matching-index access. Rows that satisfy the compound predicate `(C1>5 OR C2=7)` are passed to stage 2 for evaluation of the predicate `C3=C4`.

`WHERE C1= 17 and C2 <> 100`

In this example, assuming that a RANDOM ordering option has been specified on C2 in the CREATE INDEX statement, the query can use the index only in a limited way. The index is an effective filter on C1, but it would not match on C2 because of the random values. The index is scanned for all values where `C1=17` and only then ensure that values for C2 are not equal to 100.

`WHERE (C1 = 1 OR C2 = 1) AND XMLEXISTS('/a/b[c = 1]' PASSING XML_COL1) AND XMLEXISTS('/a/b[(e = 2 or f[g] = 3) and /h/i[j] = 4]' PASSING XML_COL2)`

The compound predicate `(C1 = 1 OR C2 = 1)` is indexable and stage 1. The first XMLEXISTS predicate is indexable and can become a matching predicate if the XML index /a/b/c has been created. The second XMLEXISTS predicate is indexable and can use multiple index access if the XML indexes, `/a/b/e`, `/a/b/f/g`, and `/a/b/h/i/j`, can be used to evaluate three XPath segments in the predicate. All rows satisfying the three indexable predicates (one compound and two XMLEXISTS) are passed to stage 2 to evaluate the same first and second XMLEXISTS predicates again.

PSPI

Related reference:

- where-clause (DB2 SQL)
- XMLEXISTS predicate (DB2 SQL)

Summary of predicate processing

You can improve performance of your SQL statements by specifying predicates that are evaluated at earlier stages.

Processing order

Predicates are applied by *stage* in the following order:

1. *Indexable* predicates that match on index key columns are applied and evaluated when the index is accessed.
2. Stage 1 *index screening* predicates that have not been picked as index matching predicates but still refer to index columns, are applied to the index.

3. Stage 1 *page range screening* predicates refer to partitioning columns are applied to limit the number of partitions that are accessed.
4. Other stage 1 predicates are applied to the data, after data page access.
5. The *stage 2* predicates are applied on the returned data rows.

The STAGE column of DSN_FILTER_TABLE indicates the stage at which a predicate was applied.

Within each stage after the indexable stage, predicates are applied in the following order, by type:

1. Equality predicates (including IN predicates that contain only a single item and BETWEEN predicates that contain the same value twice)
2. Range predicates and predicates of the form *column* IS NOT NULL
3. Other predicate types

After both sets of rules are applied, predicates are evaluated in the order in which they appear in the query. Because you specify that order, you have some control over the order of evaluation. However, Regardless of coding order, non-correlated subqueries are evaluated before correlated subqueries, unless DB2 correlates, de-correlates, or transforms the subquery into a join.

Predicate types and processing by stage

PSPI

In general, if you form a compound predicate by combining several simple predicates with OR operators, the result of the operation has the same characteristics as the simple predicate that is evaluated latest. For example, if two indexable predicates are combined with an OR operator, the result is indexable. If a stage 1 predicate and a stage 2 predicate are combined with an OR operator, the result is stage 2.

Indexable and stage 1 predicates [31]

The following predicates might be evaluated by matching index access, during index screening, or after data page access during stage 1 processing.

- COL = *value* [17, 31]
- COL = *noncol expr* [9, 11, 12, 15, 16, 30, 31]
- COL IS NULL [21, 22]
- COL *op value* [13, 31]
- COL *op noncol expr* [9, 11, 12, 13, 30, 31]
- COL BETWEEN *value1* AND *value2* [13]
- COL BETWEEN *noncol expr 1* AND *noncol expr 2* [9, 11, 12, 13, 24, 30]
- COL BETWEEN *expr-1* AND *expr-2* [6, 7, 11, 12, 13, 14, 15, 16, 28, 30]
- COL LIKE '*pattern*' [30]
- COL IN (*list*) [18, 19]
- COL IS NOT NULL [22]
- COL LIKE *host variable* [2, 30]
- COL LIKE UPPER ('*pattern*') [30]
- COL LIKE UPPER (*host-variable*) [2, 30]
- COL LIKE UPPER (*SQL-variable*)[2, 30]
- COL LIKE UPPER (CAST ('*pattern*' AS *data-type*))[2, 30]

- COL LIKE UPPER (CAST (*host-variable* AS *data-type*))[2, 30]
- COL LIKE UPPER (CAST (*SQL-variable* AS *data-type*))[2, 30]
- [2, 30]
- T1.COL = T2.COL
- T1.COL *op* T2.COL
- T1.COL = T2 *col expr* [6, 9, 11, 12, 14, 15,16, 26, 28, 30]
- T1.COL *op* T2 *col expr* [6, 9, 11, 12, 13, 14, 15, 16, 30]
- COL = (*noncor subq*)
- COL *op* (*noncor subq*) [29]
- COL = ANY (*noncor subq*) [23, 30]
- (COL1,...COL*n*) IN (*noncor subq*) [30]
- COL = ANY (*cor subq*) [20, 23, 30]
- COL IS NOT DISTINCT FROM *value* [17]
- COL IS NOT DISTINCT FROM *noncol expr* [9, 11, 12, 15,16, 30]
- T1.COL1 IS NOT DISTINCT FROM T2.COL2 [3, 4]
- T1.COL1 IS NOT DISTINCT FROM T2 *col expr* [6, 9, 11, 12, 14, 15,16, 30]
- COL IS NOT DISTINCT FROM (*noncor subq*)

Stage 1 not indexable predicates [31]

The following predicates might be evaluated during stage 1 processing, during index screening, or after data page access.

- COL <> *value* [8, 11]
- COL <> *noncol expr* [8, 11, 30]
- COL NOT BETWEEN *value1* AND *value2*
- COL NOT IN (*list*)
- COL NOT LIKE ' *char*' [30]
- COL LIKE '%*char*' [1, 30]
- COL LIKE '_*char*' [1, 30]
- T1.COL <> T2 *col expr* [8, 11, 28, 30]
- COL *op* ANY (*noncor subq*) [23]
- COL *op* ALL (*noncor subq*)
- COL IS DISTINCT FROM *value* [8, 11]
- COL IS DISTINCT FROM (*noncor subq*)

Stage 2 predicates

The following predicates must be processed during stage 2, after the data is returned.

- *value* BETWEEN COL1 AND COL2
- COL BETWEEN COL1 AND COL2 [10]
- *value* NOT BETWEEN COL1 AND COL2
- *value* BETWEEN *col expr* and *col expr*
- T1.COL <> T2.COL
- T1.COL1 = T1.COL2 [3,26]
- T1.COL1 *op* T1.COL2 [3]
- T1.COL1 <> T1.COL2 [3]
- COL = ALL (*noncor subq*)
- COL <> (*noncor subq*) [23]

- COL <> ALL (*noncor subq*)
- COL NOT IN (*noncor subq*)
- COL = (*cor subq*) [5]
- COL = ALL (*cor subq*)
- COL *op* (*cor subq*) [5]
- COL *op* ANY (*cor subq*) [23]
- COL *op* ALL (*cor subq*)
- COL <> (*cor subq*) [5]
- COL <> ANY (*cor subq*) [20]
- (COL1,...COL*n*) IN (*cor subq*)
- COL NOT IN (*cor subq*)
- (COL1,...COL*n*) NOT IN (*cor subq*)
- T1.COL1 IS DISTINCT FROM T2.COL2 [3]
- T1.COL1 IS DISTINCT FROM T2 *col expr* [8, 11]
- COL IS NOT DISTINCT FROM (*cor subq*)
- EXISTS (*subq*)[20]
- *expression* = *value* [28]
- *expression* <> *value* [28]
- *expression op value* [28]
- *expression op* (*subq*)
- NOT XMLEXISTS

Indexable but not stage 1 predicates

The following predicates can be processed during index access, but cannot be processed during stage 1.

- XMLEXISTS [27]

Notes:

1. Indexable only if an ESCAPE character is specified and used in the LIKE predicate. For example, COL LIKE '+%*char*' ESCAPE '+' is indexable.
2. Indexable only if the pattern in the variable is an indexable constant (for example, variable='*char*%').
3. If both COL1 and COL2 are from the same table, access through an index on either one is not considered for these predicates. However, the following query is an exception:

   ```
   SELECT *  FROM T1 A, T1 B WHERE A.C1 = B.C2;
   ```

   By using correlation names, the query treats one table as if it were two separate tables. Therefore, indexes on columns C1 and C2 are considered for access.
4. The predicate might be indexable and stage 1, if both sides contain the same data type. Otherwise, the predicate is stage 2.
5. If the subquery has already been evaluated for a given correlation value, then the subquery might not have to be reevaluated.
6. The column on the left side of the join sequence must be in a different table from any columns on the right side of the join sequence.
7. The tables that contain the columns in *expression1* or *expression2* must already have been accessed.

8. The processing for WHERE NOT COL = *value* is like that for WHERE COL <> *value*, and so on.
9. If *noncol expr*, *noncol expr1*, or *noncol expr2* is a noncolumn expression of one of these forms, then the predicate is not indexable:
   - *noncol expr* + 0
   - *noncol expr* - 0
   - *noncol expr* * 1
   - *noncol expr* / 1
   - *noncol expr* CONCAT *empty string*
10. COL, COL1, and COL2 can be the same column or different columns. The columns are in the same table.
11. Any of the following sets of conditions make the predicate stage 2:
    - The first value obtained before the predicate is evaluated is BIGINT or DECIMAL(*p*,*s*), where *p*>15, and the second value obtained before the predicate is evaluated is REAL or FLOAT.
    - The first value obtained before the predicate is evaluated is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC, and the second value obtained before the predicate is evaluated is DATE, TIME, or TIMESTAMP.
12. The predicate is stage 1 but not indexable if the first value obtained before the predicate is evaluated is CHAR or VARCHAR, the second value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC, and the first value obtained before the predicate is evaluated is not Unicode mixed.
13. If both sides of the comparison are strings, any of the following sets of conditions makes the predicate stage 1 but not indexable:
    - The first value obtained before the predicate is evaluated is CHAR or VARCHAR, and the second value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC.
    - Both of the following conditions are true:
      - Both sides of the comparison are CHAR or VARCHAR, or both sides of the comparison are BINARY or VARBINARY
      - The length the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
    - Both of the following conditions are true:
      - Both sides of the comparison are GRAPHIC or VARGRAPHIC.
      - The length of the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
    - Both of the following conditions are true:
      - The first value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC, and the second value obtained before the predicate is evaluated is CHAR or VARCHAR.
      - The length of the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
14. If both sides of the comparison are strings, but the two sides have different CCSIDs, the predicate is stage 1 and indexable only if the first value obtained before the predicate is evaluated is Unicode and the comparison does not meet any of the conditions in note 13.
15. If *col expr* or *noncol expr* is a CASE expression, the predicate is stage 2.

16. If all of the following conditions are true, the predicate is stage 2:
    - *col expr* or *noncol expr* is an integer value that is the product or the quotient of two non-column expressions
    - COL is a FLOAT or a DECIMAL column
17. If COL has the ROWID data type, DB2 tries to use direct row access instead of index access or a table space scan.
18. If COL has the ROWID data type, and an index is defined on COL, DB2 tries to use direct row access instead of index access.
19. IN-list predicates are indexable and stage 1 if the following conditions are true:
    - The IN list contains only simple items. For example, constants, host variables, parameter markers, and special registers.
    - The IN list does not contain any aggregate functions or scalar functions.
    - The IN list is not contained in a trigger's WHEN clause.
    - For numeric predicates where the left side column is DECIMAL with precision greater than 15, none of the items in the IN list are FLOAT.
    - For string predicates, the coded character set identifier is the same as the identifier for the left side column.
    - For DATE, TIME, and TIMESTAMP predicates, the left side column must be DATE, TIME, or TIMESTAMP.
20. Certain predicates might become indexable and stage 1 depending on how they are transformed during processing.
21. The predicate types COL IS NULL and COL IS NOT NULL are stage 2 predicates when they query a column that is defined as NOT NULL.
22. If the predicate type is COL IS NULL and the column is defined as NOT NULL, the table is not accessed because C1 cannot be NULL.
23. The ANY and SOME keywords behave similarly. If a predicate with the ANY keyword is not indexable and not stage 1, a similar predicate with the SOME keyword is not indexable and not stage 1.
24. Under either of these circumstances, the predicate is stage 2:
    - *noncol expr* is a case expression.
    - *noncol expr* is the product or the quotient of two noncolumn expressions, that product or quotient is an integer value, and COL is a FLOAT or a DECIMAL column.
25. COL IN (*noncor subq*) is stage 1 for type N access only. Otherwise, it is stage 2.
26. If the inner table is an EBCDIC or ASCII column and the outer table is a Unicode column, the predicate is stage 1 and indexable.
27. The XMLEXISTS is always stage 2. But the same predicate can be indexable and become the matching predicate if an XML index can be used to evaluate the XPath expression in the predicate. The XMLEXISTS predicate can never be a screening predicate.
28. The predicate might be indexable by an expression-based index if it contains an expression that is a column reference, invokes a built-in function, or contains a general expression.
29. This type of predicate is not stage 1 when a nullability mismatch is possible.
30. If COL is defined with a field procedure, the predicate becomes stage 2.
31. The following predicates might be indexable and stage 1 if only the right side contains a DECFLOAT data type:
    - `COL = *value*`

- COL = *noncol expr*
- COL *op value*
- COL *op noncol expr*

Other predicates that contain a DECFLOAT data type are not stage 1 and not indexable.

32. COL is a DATE, TIME, or TIMESTAMP column.

The following abbreviations and variable values are used in the preceding in the sample predicates:

char Any character string that does not include the special characters for percent (%) or underscore (_).

COL A column name.

col expr
A column expression.

cor subq
A correlated subquery

expression
Any expression that contains arithmetic operators, scalar functions, aggregate functions, concatenation operators, columns, constants, host variables, special registers, or date or time expressions.

noncol expr
A non-column expression, which is any expression that does not contain a column. That expression can contain arithmetic operators, scalar functions, concatenation operators, constants, host variables, special registers, or date or time expressions.

An example of a non-column expression is

```
CURRENT DATE - 50 DAYS
```

noncor subq
A non-correlated subquery

op any of the operators >, >=, <, <=, ¬>, ¬<

predicate
A predicate of any type.

pattern
Any character string that does *not* start with the special characters for percent (%) or underscore (_).

subq A correlated or noncorrelated subquery

T*n* A table name.

T*n* *col expr*
An expression that contains a column in table T*n*. The expression might be only that column.

value A constant, host variable, or special register.

PSPI

Related concepts:

Stage 1 and stage 2 predicates

Indexable and non-indexable predicates

Related reference:

Examples of predicate properties

DSN_FILTER_TABLE

Ensuring that predicates are coded correctly

Whether you code the predicates of your SQL statements correctly has a great effect on the performance of those queries.

Procedure

PSPI

To ensure the best performance from your predicates:

- Make sure all the predicates that you think should be indexable are coded so that they can be indexable. Refer to "Predicate types and processing by stage" on page 345 to see which predicates are indexable and which are not.
- Try to remove any predicates that are unintentionally redundant or not needed; they can slow down performance.
- For string comparisons other than equal comparisons, ensure that the declared length of a host variable is less than or equal to the length attribute of the table column that it is compared to. For languages in which character strings are null-terminated, the string length can be less than or equal to the column length plus 1. If the declared length of the host variable is greater than the column length,in a non-equal comparison, the predicate is stage 1 but cannot be a matching predicate for an index scan.

  For example, assume that a host variable and an SQL column are defined as follows:

C language declaration	SQL definition
char string_hv[15]	STRING_COL CHAR(12)

  A predicate such as `WHERE STRING_COL > :string_hv` is not a matching predicate for an index scan because the length of string_hv is greater than the length of STRING_COL. One way to avoid an inefficient predicate using character host variables is to declare the host variable with a length that is less than or equal to the column length:

```
char string_hv[12]
```

  Because this is a C language example, the host variable length could be 1 byte greater than the column length:

```
char string_hv[13]
```

  For numeric comparisons, a comparison between a DECIMAL column and a float or real host variable is stage 2 if the precision of the DECIMAL column is greater than 15. For example, assume that a host variable and an SQL column are defined as follows:

C language declaration	SQL definition
float float_hv	DECIMAL_COL DECIMAL(16,2)

  A predicate such as `WHERE DECIMAL_COL = :float_hv` is not a matching predicate for an index scan because the length of DECIMAL_COL is greater than 15. However, if DECIMAL_COL is defined as DECIMAL(15,2), the predicate is stage 1 and indexable.

PSPI

Predicate filter factors

By understanding of how DB2 uses filter factors you can write more efficient predicates.

PSPI

The *filter factor* of a predicate is a number 0 - 1. The number estimates the proportion of rows in a table for which the predicate is true. Those rows are said to *qualify* by that predicate.

For example, suppose that DB2 can determine that column C1 of table T contains only five distinct values: A, D, Q, W, and X. In the absence of other information, DB2 estimates that one-fifth of the rows contain the value D in column C1. Then the predicate `C1='D'` has the filter factor 0.2 for table T.

How DB2 uses filter factors:

DB2 uses filter factors to estimate the number of rows that are qualified by a set of predicates.

For simple predicates, the filter factor is a function of three variables:

- The constant value in the predicate; for instance, 'D' in the previous example.
- The operator in the predicate; for instance, '=' in the previous example and '<>' in the negation of the predicate.
- Statistics on the column in the predicate. In the previous example, an example statistics is the information that column T.C1 contains only five values.

Tip: You can control the first two of those variables when you write a predicate. Your understanding of how DB2 uses filter factors can help you to write more efficient predicates.

Values of the third variable, statistics on the column, are kept in the DB2 catalog. You can update many of those values by running the RUNSTATS utility. You can also modify the catalog table values.

Important: Before you modify the catalog with statistics of your own choice, you must understand how DB2 uses filter factors and interpolation formulas.

PSPI

Related tasks:

Maintaining DB2 database statistics

Modifying catalog statistics to influence access path selection

Related reference:

RUNSTATS (DB2 Utilities)

Statistics used for access path selection

Default filter factors for simple predicates

DB2 uses default filter factor values when no other statistics exist.

PSPI

The following table lists default filter factors for different types of predicates.

Table 62. DB2 default filter factors by predicate type

Predicate Type	**Filter Factor**
Col = *constant*	0.04
Col <> *constant*	0.96
Col IS NULL	0.04
Col IS NOT DISTINCT FROM	0.04
Col IS DISTINCT FROM	0.96
Col IN (*constant-list*)	MIN(*number-of-constants*/25, 1.0)
Col *Op constant*, where *Op* is one of these operators: <, <=, >, >=.	0.33
Col LIKE *constant*	0.10
Col BETWEEN *constant1* and *constant2*	0.10

Example

The default filter factor for the predicate C1 = 'D' is 1/25 (0.04). However, If the selectivity of the D value is actually not close to 0.04, the default filter factor probably does not lead to an optimal access path. In such cases, statistics might be needed to improve access path selection.

Filter factors for other predicate types:

Examples above represent only the most common types of predicates. If P1 is a predicate and F is its filter factor, then the filter factor of the predicate NOT P1 is (1 - F). But, filter factor calculation is dependent on many things, so a specific filter factor cannot be given for all predicate types.

PSPI

Related tasks:

Maintaining DB2 database statistics

Improving filter factors by collecting cardinality and frequency statistics

Related reference:

Statistics used for access path selection

Filter factors for uniform distributions

In certain situations DB2 assumes that a data is distributed uniformly and calculates filter factors accordingly.

PSPI

DB2 uses the filter factors in the following table if the following conditions are true:

- The value in column COLCARDF of catalog table SYSIBM.SYSCOLUMNS for the column “Col” is a positive value.
- No additional statistics exist for “Col” in SYSIBM.SYSCOLDIST.

Table 63. DB2 uniform filter factors by predicate type

Predicate type	**Filter factor**
Col = *constant*	1/COLCARDF
Col <> *constant*	1 – (1/COLCARDF)
Col IS NULL	1/COLCARDF
Col IS NOT DISTINCT FROM	1/COLCARDF
Col IS DISTINCT FROM	1 – (1/COLCARDF)
Col IN (constant list)	*number of constants* / COLCARDF
Col *Op1*[1] constant	interpolation formula
Col *Op2*[2] constant	interpolation formula
Col LIKE constant	interpolation formula
Col BETWEEN *constant1* and *constant2*	interpolation formula

Notes:

1. *Op1* is < or <=, and the constant is not a host variable.
2. *Op2* is > or >=, and the constant is not a host variable.

Example

If D is one of only five values in column C1, using RUNSTATS puts the value 5 in column COLCARDF of SYSCOLUMNS. If no additional statistics are available, the filter factor for the predicate C1 = 'D' is 1/5 (0.2).

Filter factors for other predicate types:

Examples above represent only the most common types of predicates. If P1 is a predicate and F is its filter factor, then the filter factor of the predicate NOT P1 is (1 - F). But, filter factor calculation is dependent on many things, so a specific filter factor cannot be given for all predicate types.

PSPI

Related tasks:

Maintaining DB2 database statistics

Improving filter factors by collecting cardinality and frequency statistics

Related reference:

Statistics used for access path selection

Interpolation formulas

For a predicate that uses a range of values, DB2 calculates the filter factor by an *interpolation formula*.

PSPI

The formula is based on an estimate of the ratio of the number of values in the range to the number of values in the entire column of the table.

The formulas

The formulas that follow are rough estimates, which are subject to further modification by DB2. They apply to a predicate of the form `col op. constant`. The value of (Total Entries) in each formula is estimated from the values in columns HIGH2KEY and LOW2KEY in catalog table SYSIBM.SYSCOLUMNS for column *col*: Total Entries = (*HIGH2KEY value - LOW2KEY value*).

- For the operators < and <=, where the constant is not a host variable:

  (*constant value - LOW2KEY value*) / (*Total Entries*)
- For the operators > and >=, where the constant is not a host variable:

  (*HIGH2KEY value - constant value*) / (*Total Entries*)
- For LIKE or BETWEEN:

  (*High constant value - Low constant value*) / (*Total Entries*)

Example

For column C2 in a predicate, suppose that the value of HIGH2KEY is 1400 and the value of LOW2KEY is 200. For C2, DB2 calculates *Total Entries* = 1400 - 200, or 1200.

For the predicate `C1 BETWEEN 800 AND 1100`, DB2 calculates the filter factor F as:

F = (1100 - 800)/1200 = 1/4 = 0.25

Interpolation for LIKE

DB2 treats a LIKE predicate as a type of BETWEEN predicate. Two values that bound the range qualified by the predicate are generated from the constant string in the predicate. Only the leading characters found before the escape character ('%' or '_') are used to generate the bounds. So if the escape character is the first character of the string, the filter factor is estimated as 1, and the predicate is estimated to reject no rows.

Defaults for interpolation

DB2 might not interpolate in some cases; instead, it can use a default filter factor. Defaults for interpolation are:

- Relevant only for ranges, including LIKE and BETWEEN predicates
- Used only when interpolation is not adequate
- Based on the value of COLCARDF
- Used whether uniform or additional distribution statistics exist on the column if either of the following conditions is met:
  - The predicate does not contain constants
  - COLCARDF < 4.

The following table shows interpolation defaults for the operators <, <=, >, >= and for LIKE and BETWEEN.

Table 64. Default filter factors for interpolation

COLCARDF	Factor for *Op*[1]	Factor for LIKE or BETWEEN
>=100000000	1/10,000	3/100000
>=10000000	1/3,000	1/10000

Table 64. Default filter factors for interpolation (continued)

COLCARDF	Factor for *Op*[1]	Factor for LIKE or BETWEEN
>=1000000	1/1,000	3/10000
>=100000	1/300	1/1000
>=10000	1/100	3/1000
>=1000	1/30	1/100
>=100	1/10	3/100
>=2	1/3	1/10
=1	1/1	1/1
<=0	1/3	1/10

Note:

1. *Op* is one of these operators: <, <=, >, >=.

PSPI

Filter factors for all distributions

RUNSTATS can generate additional statistics for a column or a group of columns. DB2 can use that information to calculate filter factors.

PSPI

DB2 collects two kinds of distribution statistics:

Frequency
The percentage of rows in the table that contain a value for a column or set of columns

Cardinality
The number of distinct values in a set of columns

The table that follows lists the types of predicates on which these statistics are used.

Table 65. Predicates for which distribution statistics are used

Type of statistic	Single or concatenated columns	Predicates
Frequency	Single	COL=*constant* COL IS NULL COL IN (*constant-list*) COL *op*[1] *constant* COL BETWEEN *constant* AND *constant* COL=*host-variable* COL1=COL2 T1.COL=T2.COL COL IS NOT DISTINCT FROM
Frequency	Concatenated	COL=*constant* COL IS NOT DISTINCT FROM

Table 65. Predicates for which distribution statistics are used (continued)

Type of statistic	Single or concatenated columns	Predicates
Cardinality	Single	COL=*constant* COL IS NULL COL IN (*constant-list*) COL *op*[1] *constant* COL BETWEEN *constant* AND *constant* COL=*host-variable* COL1=COL2 T1.COL=T2.COL COL IS NOT DISTINCT FROM

Cardinality	Concatenated	COL=*constant*
		COL=*:host-variable*
		COL1=COL2
		COL IS NOT DISTINCT FROM

Note:

1. *op* is one of these operators: <, <=, >, >=.

How DB2 uses frequency statistics

Columns COLVALUE and FREQUENCYF in table SYSCOLDIST contain distribution statistics. Regardless of the number of values in those columns, running RUNSTATS deletes the existing values and inserts rows for frequent values.

You can run RUNSTATS without the FREQVAL option, with the FREQVAL option in the *correl-spec*, with the FREQVAL option in the *colgroup-spec*, or in both, with the following effects:

- If you run RUNSTATS without the FREQVAL option, RUNSTATS inserts rows for the 10 most frequent values for the first column of the specified index.
- If you run RUNSTATS with the FREQVAL option in the *correl-spec*, RUNSTATS inserts rows for concatenated columns of an index. The NUMCOLS option specifies the number of concatenated index columns. The COUNT option specifies the number of frequent values. You can collect most-frequent values, least-frequent values, or both.
- If you run RUNSTATS with the FREQVAL option in the *colgroup-spec*, RUNSTATS inserts rows for the columns in the column group that you specify. The COUNT option specifies the number of frequent values. You can collect most-frequent values, least-frequent values, or both.
- If you specify the FREQVAL option, RUNSTATS inserts rows for columns of the specified index and for columns in a column group.

DB2 uses the frequencies in column FREQUENCYF for predicates that use the values in column COLVALUE and assumes that the remaining data are uniformly distributed.

Example: Filter factor for a single column

Suppose that the predicate is C1 IN ('3','5') and that SYSCOLDIST contains these values for column C1:

```
  COLVALUE    FREQUENCYF
    '3'         .0153
    '5'         .0859
    '8'         .0627
```

The filter factor is .0153 + .0859 = .1012.

Example: Filter factor for correlated columns

Suppose that columns C1 and C2 are correlated. Suppose also that the predicate is C1='3' AND C2='5' and that SYSCOLDIST contains these values for columns C1 and C2:

```
COLVALUE    FREQUENCYF
'1' '1'       .1176
'2' '2'       .0588
'3' '3'       .0588
'3' '5'       .1176
'4' '4'       .0588
'5' '3'       .1764
'5' '5'       .3529
'6' '6'       .0588
```

The filter factor is .1176.

PSPI

Histogram statistics filter factors

When histogram statistics are available, DB2 can more accurately interpolate the distribution of values across a large range.

PSPI

RUNSTATS normally collects frequency statistics for a single-column or single multi-column data set. Because catalog space and bind time performance concerns make the collection of these types of statistics on every distinct value found in the target column or columns very impractical, such frequency statistics are commonly collected only on the most frequent or least frequent, and therefore most biased, values.

However, such limited statistics often do not provide an accurate prediction of the value distribution because they require a rough interpolation across the entire range of values. For example, suppose that the YRS_OF_EXPERIENCE column on an EMPLOYEE table contains the following value frequencies:

Table 66. Example frequency statistics for values on the YRS_OF_EXPERIENCE column in an EMPLOYEE table

VALUE	FREQUENCY
2	10%
25	15%
26	15%
27	7%
12	0.02%
13	0.01%
40	0.0001%

Table 66. Example frequency statistics for values on the YRS_OF_EXPERIENCE column in an EMPLOYEE table (continued)

VALUE	FREQUENCY
41	0.00001%

Example predicates that can benefit from histogram statistics

Some example predicates on values in this table include:

- Equality predicate with unmatched value:

```
SELECT EMPID FROM EMPLOYEE T
WHERE T.YRS_OF_EXPERIENCE = 6;
```

- Range predicate:

```
SELECT T.EMPID FROM EMPLOYEE T
WHERE T.YRS_OF_EXPERIENCE BETWEEN 5 AND 10;
```

- Non-local predicate:

```
SELECT T1.EMPID FROM EMPLOYEE T1, OPENJOBS T2
WHERE T1.SPECIALTY = T2.AREA AND T1.YRS_OF_EXPERIENCE > T2.YRS_OF_EXPERIENCE;
```

For each of the above predicates, distribution statistics for any single value cannot help DB2 to estimate predicate selectivity, other than by uniform interpolation of filter factors over the uncollected part of the value range. The result of such interpolation might lead to inaccurate estimation and undesirable access path selection.

How DB2 uses histogram statistics

DB2 creates a number of intervals such that each interval contains approximately the same percentage of rows from the data set. The number of intervals is specified by the value of NUMQUANTILES when you use the HISTOGRAM option of RUNSTATS. Each interval has an identifier value QUANTILENO, and values, the LOWVALUE and HIGHVALUE columns, that bound the interval. DB2 collects distribution statistics for each interval.

When you use RUNSTATS to collect statistics on a column that contains such wide-ranging frequency values, specify the HISTORGRAM option to collect more granular distribution statistics that account for the distribution of values across the entire range of values. The following table shows the result of collecting histogram statistics for the years of experience values in the employee table. In this example, the statistics have been collected with 7 intervals:

Table 67. Histogram statistics for the column YRS_OF_EXPERIENCE in an EMPLOYEE table.

QUANTILENO	LOWVALUE	HIGHVALUE	CARDF	FREQUENCYF
1	0	3	4	14%
2	4	15	8	14%
3	18	24	7	12%
4	25	25	1	15%
5	26	26	1	15%
6	27	30	4	16%
7	35	40	6	14%

PSPI

Related concepts:

Histogram statistics

Related tasks:

Collecting histogram statistics

Related reference:

RUNSTATS (DB2 Utilities)

SYSIBM.SYSCOLDIST table (DB2 SQL)

SYSIBM.SYSKEYTGTDIST table (DB2 SQL)

Related information:

Histogram statistics over a range of column values (DB2 9 for z/OS Performance Topics)

Histogram statistics (DB2 9 for z/OS Performance Topics)

Histogram statistics recommendations (DB2 9 for z/OS Performance Topics)

How DB2 uses multiple filter factors to determine the cost of a query

When DB2 estimates the cost of a query, it determines the filter factor repeatedly and at various levels.

PSPI

For example, suppose that you execute the following query:

```
SELECT COLS FROM T1
  WHERE C1 = 'A'
  AND   C3 = 'B'
  AND   C4 = 'C';
```

Table T1 consists of columns C1, C2, C3, and C4. Index I1 is defined on table T1 and contains columns C1, C2, and C3.

Suppose that the simple predicates in the compound predicate have the following characteristics:

C1='A'
Matching predicate

C3='B' Screening predicate

C4='C' Stage 1, nonindexable predicate

To determine the cost of accessing table T1 through index I1, DB2 performs these steps:

1. Estimates the matching index cost. DB2 determines the index matching filter factor by using single-column cardinality and single-column frequency statistics because only one column can be a matching column.
2. Estimates the total index filtering. This includes matching and screening filtering. If statistics exist on column group (C1,C3), DB2 uses those statistics. Otherwise DB2 uses the available single-column statistics for each of these columns.

DB2 also uses FULLKEYCARDF as a bound. Therefore, it can be critical to have column group statistics on column group (C1, C3) to get an accurate estimate.

3. Estimates the table-level filtering. If statistics are available on column group (C1,C3,C4), DB2 uses them. Otherwise, DB2 uses statistics that exist on subsets of those columns.

Important: If you supply appropriate statistics at each level of filtering, DB2 is more likely to choose the most efficient access path.

You can use RUNSTATS to collect any of the needed statistics.

PSPI

Filter factor estimation for the XMLEXISTS predicate

DB2 uses available statistics for implicit and explicit XML objects to estimate the filter factors for XMLEXISTS predicates.

PSPI

If the preceding conditions are not true, DB2 calculates the filter factor for the XPath predicates by applying a formula that uses the following values:

- The filter factor of the auxiliary table. The filter factors of all XPath predicates in the XMLEXISTS predicate indicate the filtering on the XML table. If the FULLKEYCARD value is available for the node ID index, it is used as the default CARDF value of the XML table.
- The FIRSKEYCARDF value for the implicitly created node ID index
- The CARDF value for the base table.

The following rules apply to these values:

- When the statistics are available for the node ID index, the FIRSTKEYCARDF value is the number of distinct DOCID values in the XML table.
- When the statistics are available for the XML index, the FIRSTKEYCARDF value is used as the COLCARD value of the comparison operands in the XPath predicates.
- When statistics are not available, the default filter factor, is the same as for non-XPath predicates that have the same comparison type.
- If the XML index can be used to evaluate the XPath predicate, the default filter factor is redefined based on the FIRSTKEYCARDF value. The filter factor is the same as non-XPath predicates with the same comparison type and the same COLCARD value.
- When no statistics are available for the node ID index or the XML index, the following default statistics are used to estimate the filter factor:
  - NLEAF = *XML-table-CARDF* / 300
  - NLEVELS uses the default value for the node ID index and the XML index for the XMLEXISTS predicate. PSPI

  Because the index statistics are not available to help the default filter factor estimation, the predicate filter factor is set according to the predicate comparison type.

Related concepts:

Additional statistics that provide index costs

Storage structure for XML data (DB2 Programming for XML)

Access methods with XML indexes (DB2 Programming for XML)

Related reference:

Statistics used for access path selection

Default filter factors for simple predicates

XMLEXISTS predicate (DB2 SQL)

Avoiding problems with correlated columns

Two columns in a table are said to be *correlated* if the values in the columns do not vary independently.

About this task

DB2 might not determine the best access path when your queries include correlated columns.

Procedure

PSPI

To address problems with correlated columns:

- For leading indexed columns, run the RUNSTATS utility. For all other column groups, run the RUNSTATS utility with the COLGROUP option
- Run the RUNSTATS utility to collect column correlation information for any column group with the COLGROUP option.
- Update the catalog statistics manually.
- Use SQL that forces access through a particular index.

Results

The RUNSTATS utility collects the statistics that DB2 needs to make proper choices about queries. With RUNSTATS, you can collect statistics on the concatenated key columns of an index and the number of distinct values for those concatenated columns. This gives DB2 accurate information to calculate the filter factor for the query.

Example

For example, RUNSTATS collects statistics that benefit queries like this:

```
SELECT * FROM T1
 WHERE C1 = 'a' AND C2 = 'b' AND C3 = 'c' ;
```

Where:

- The first three index keys are used (MATCHCOLS = 3).
- An index exists on C1, C2, C3, C4, C5.
- Some or all of the columns in the index are correlated in some way.

PSPI

Related tasks:

Maintaining DB2 database statistics

Modifying catalog statistics to influence access path selection

Related reference:

RUNSTATS (DB2 Utilities)

Correlated columns

Two columns of data, A and B of a single table, are *correlated* if the values in column A do not vary independently of the values in column B.

PSPI

For example, the following table is an excerpt from a large single table. Columns CITY and STATE are highly correlated, and columns DEPTNO and SEX are entirely independent.

Table 68. Sample data from the CREWINFO table

CITY	STATE	DEPTNO	SEX	EMPNO	ZIPCODE
Fresno	CA	A345	F	27375	93650
Fresno	CA	J123	M	12345	93710
Fresno	CA	J123	F	93875	93650
Fresno	CA	J123	F	52325	93792
New York	NY	J123	M	19823	09001
New York	NY	A345	M	15522	09530
Miami	FL	B499	M	83825	33116
Miami	FL	A345	F	35785	34099
Los Angeles	CA	X987	M	12131	90077
Los Angeles	CA	A345	M	38251	90091

In this simple example, for every value of column CITY that equals 'FRESNO', the STATE column contains the same value ('CA').

PSPI

Impacts of correlated columns

DB2 might not determine the best access path, table order, or join method when your query uses columns that are highly correlated.

PSPI

Column correlation can make the estimated cost of operations cheaper than they actually are. Correlated columns affect both single-table queries and join queries.

Column correlation on the best matching columns of an index

The following query selects rows with females in department A345 from Fresno, California. Two indexes are defined on the table, Index 1 (CITY,STATE,ZIPCODE) and Index 2 (DEPTNO,SEX).

Query 1

```
SELECT ... FROM CREWINFO WHERE
  CITY = 'FRESNO' AND STATE = 'CA'         (PREDICATE1)
  AND DEPTNO = 'A345' AND SEX = 'F';       (PREDICATE2)
```

Consider the two compound predicates (labeled PREDICATE1 and PREDICATE2), their actual filtering effects (the proportion of rows they select), and their DB2 filter factors. Unless the proper catalog statistics are gathered, the filter factors are calculated as if the columns of the predicate are entirely independent (not correlated).

When the columns in a predicate correlate but the correlation is not reflected in catalog statistics, the actual filtering effect to be significantly different from the DB2 filter factor. The following table shows how the actual filtering effect and the DB2 filter factor can differ, and how that difference can affect index choice and performance.

Table 69. Effects of column correlation on matching columns

	INDEX 1	**INDEX 2**
Matching predicates	Predicate1 CITY=FRESNO AND STATE=CA	Predicate2 DEPTNO=A345 AND SEX=F
Matching columns	2	2

DB2 estimate for matching columns (Filter Factor)	column=CITY, COLCARDF=4
	Filter Factor=1/4
	column=STATE, COLCARDF=3
	Filter Factor=1/3
		column=DEPTNO, COLCARDF=4
		Filter Factor=1/4
		column=SEX, COLCARDF=2
		Filter Factor=1/2

Compound filter factor for matching columns	1/4 × 1/3 = 0.083	1/4 × 1/2 = 0.125
Qualified leaf pages based on DB2 estimations	0.083 × 10 = 0.83 INDEX CHOSEN (.8 < 1.25)	0.125 × 10 = 1.25
Actual filter factor based on data distribution	4/10	2/10
Actual number of qualified leaf pages based on compound predicate	4/10 × 10 = 4	2/10 × 10 = 2 BETTER INDEX CHOICE (2 < 4)

DB2 chooses an index that returns the fewest rows, partly determined by the smallest filter factor of the matching columns. Assume that filter factor is the only influence on the access path. The combined filtering of columns CITY and STATE seems very good, whereas the matching columns for the second index do not seem to filter as much. Based on those calculations, DB2 chooses Index 1 as an access path for Query 1.

The problem is that the filtering of columns CITY and STATE should not look good. Column STATE does almost no filtering. Since columns DEPTNO and SEX do a better job of filtering out rows, DB2 should favor Index 2 over Index 1.

Column correlation on index screening columns of an index

Correlation might also occur on nonmatching index columns, used for index screening. See “Nonmatching index scan (ACCESSTYPE='I' and MATCHCOLS=0)” on page 714 on page 714 for more information. Index screening predicates help reduce the number of data rows that qualify while scanning the index. However, if the index screening predicates are correlated, they do not filter as many data rows as their filter factors suggest. To illustrate this, use Query 1 with the following indexes on Table 68 on page 363:

```
Index 3 (EMPNO,CITY,STATE)
Index 4 (EMPNO,DEPTNO,SEX)
```

In the case of Index 3, because the columns CITY and STATE of Predicate 1 are correlated, the index access is not improved as much as estimated by the screening predicates and therefore Index 4 might be a better choice. (Note that index screening also occurs for indexes with matching columns greater than zero.)

Multiple table joins

In Query 2, the data shown in the following table is added to the original query (see Query 1) to show the impact of column correlation on join queries.

Table 70. Data from the DEPTINFO table

CITY	STATE	MANAGER	DEPT	DEPTNAME
Fresno	CA	Smith	J123	ADMIN
Los Angeles	CA	Jones	A345	LEGAL

Query 2

```
SELECT ... FROM CREWINFO T1,DEPTINFO T2
   WHERE T1.CITY = 'FRESNO' AND T1.STATE='CA'          (PREDICATE 1)
   AND T1.DEPTNO = T2.DEPT AND T2.DEPTNAME = 'LEGAL';
```

The order that tables are accessed in a join statement affects performance. The estimated combined filtering of Predicate1 is lower than its actual filtering. So table CREWINFO might look better as the first table accessed than it should.

Also, due to the smaller estimated size for table CREWINFO, a nested loop join might be chosen for the join method. But, if many rows are selected from table CREWINFO because Predicate1 does not filter as many rows as estimated, then another join method or join sequence might be better.

PSPI

Detecting correlated columns

The first indication that correlated columns might be a problem is poor response times when DB2 has chosen an inappropriate access path. If you suspect that you have problems with correlated columns, you can issue SQL statements to test whether columns are correlated.

Procedure

To determine whether columns are correlated:

Issue SQL statements and compare the results as shown in the following example.

Example

PSPI

If you suspect two columns in a table, such as the CITY and STATE columns in the CREWINFO table might be correlated, then you can issue the following SQL queries that reflect the relationships between the columns:

```
SELECT COUNT (DISTINCT CITY) AS CITYCOUNT,
  COUNT (DISTINCT STATE) AS STATECOUNT FROM CREWINFO;
```

The result of the count of each distinct column is the value of COLCARDF in the DB2 catalog table SYSCOLUMNS. Multiply the previous two values together to get a preliminary result:

```
CITYCOUNT  x  STATECOUNT    =      ANSWER1
```

Then issue the following SQL statement:

```
SELECT COUNT(*) FROM
  (SELECT DISTINCT CITY, STATE
   FROM CREWINFO) AS V1;           (ANSWER2)
```

Compare the result of the previous count (ANSWER2) with ANSWER1. If ANSWER2 is less than ANSWER1, then the suspected columns are correlated.

PSPI

Adding extra predicates to improve access paths

By adding extra predicates to a query, you might enable DB2 to take advantage of more efficient access paths.

About this task

PSPI

DB2 performs predicate transitive closure only on equal and range predicates. However, you can help DB2 to choose a better access path by adding transitive closure predicates for other types of operators, such as LIKE.

Example

For example, consider the following SELECT statement:

```
SELECT * FROM T1,T2
   WHERE T1.C1=T2.C1
     AND T1.C1 LIKE 'A
```

If `T1.C1=T2.C1` is true, and `T1.C1 LIKE 'A%'` is true, then `T2.C1 LIKE 'A%'` must also be true. Therefore, you can give DB2 extra information for evaluating the query by adding `T2.C1 LIKE 'A%'`:

```
SELECT * FROM T1,T2
   WHERE T1.C1=T2.C1
     AND T1.C1 LIKE 'A%'
     AND T2.C1 LIKE 'A
```

PSPI

Related concepts:

Predicates generated through transitive closure

Predicate manipulation

Predicates and access path selection

Related reference:

Summary of predicate processing

Predicates for special uses

You can use certain special predicates to influence access path selection for a query.

For example, you can use the following special predicates:

`WHERE (COL1=? OR 0=1)`
The always-false `OR 0=1` condition has the following effects on the compound predicate:
- Disables index access.
- Adds 0.04 to the estimated filter factor.
- Converts it to a non-Boolean predicate and disables certain query transformations, such as pushdown and transitive closure.

`WHERE (COL1=? OR 0<>0)`
The always-false `OR 0<>0` condition has the following effects on the compound predicate:
- Disables index access.
- Adds 0.96 to the estimated filter factor, which indicates the predicate has low selectivity.
- Converts it to a non-Boolean predicate and disables certain query transformations, such as pushdown and transitive closure.

`WHERE COL1=?+0`
The `+0` is removed when the query is transformed, however the predicate is marked as not matching indexable.

`noncol-expr``+0`
`noncol-expr``-0`
`noncol-expr``*1`
`noncol-expr``/1`
`noncol-expr` **`CONCAT`** *`blank-value`*
These predicates disable index access.

Related concepts:

Indexable and non-indexable predicates

Predicate filter factors

Removal of pre-evaluated predicates

Predicates generated through transitive closure

Related reference:

Summary of predicate processing

Predicate manipulation

DB2 sometimes modifies existing predicates, generates extra predicates, or removes unneeded predicates to improve the chances that DB2 can select an efficient access path.

PSPI

The goal of such transformations is most often to increase the likelihood that DB2 can select index-based access paths, which are often more efficient than other access methods.

In most cases, you do not have to do anything to take advantage of such predicate manipulations. However, you might see the transformed predicates in EXPLAIN data in DSN_PREDICAT_TABLE, and if you view formatted query text with a query tuning tools. You might also see access path changes that result from the predicate changes in PLAN_TABLE data and in access path diagrams.

PSPI

Related concepts:

Query transformations

Interpreting data access by using EXPLAIN

Related tasks:

Generating visual representations of access plans (IBM Data Studio)

Formatting SQL statements (IBM Data Studio)

Related reference:

Summary of predicate processing

PLAN_TABLE

DSN_PREDICAT_TABLE

How DB2 modifies IN predicates

DB2 automatically modifies some queries that contain IN predicates to enable more access path options.

PSPI

DB2 converts an IN predicate that has only one item in the IN to an equality predicate. A set of simple, Boolean term, equal predicates on the same column that are connected by OR predicates can be converted into an IN predicate. For example: DB2 converts the predicate `C1=5 or C1=10 or C1=15` to `C1 IN (5,10,15)`.

DB2 also generates additional predicates through transitive closure for queries that contain IN predicates, under the following conditions:

- The INLISTP subsystem parameter is set to a positive value.
- The number of elements in the list on the right side of the IN keyword is not greater than the value of the INLISTP subsystem parameter.
- No correlated column references exist in expressions on the left side of the IN keyword.
- No columns defined by field procedures are referenced on the left side of the IN keyword
- The values in the list on the right side of the IN keyword must be simple constants. For example, the list must not include expressions involving constants, parameter markers, host variables, or subselects.
-

PSPI

Related concepts:

Query transformations

How DB2 modifies IN predicates

Predicates generated through transitive closure

Related reference:

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

How DB2 simplifies join operations

DB2 can sometimes simplify join operations to improve access path efficiency.

Introductory concepts

Ways to join data from more than one table (Introduction to DB2 for z/OS)

PSPI

However, because full outer joins are less efficient than left or right joins, and left and right joins are less efficient than inner joins, the recommendations is to always try to use the simplest type of join operation in your queries.

Simplification for predicates that eliminate null values

DB2 can simplify a join operation when the query contains a predicate or an ON clause that eliminates the null values that are generated by the join operation.

ON clauses that eliminate null values

Consider the following query:

```
SELECT * FROM T1 X FULL JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2 > 12;
```

The outer join operation yields these result table rows:

- The rows with matching values of C1 in tables T1 and T2 (the inner join result)
- The rows from T1 where C1 has no corresponding value in T2
- The rows from T2 where C1 has no corresponding value in T1

However, when you apply the predicate, you remove all rows in the result table that came from T2 where C1 has no corresponding value in T1. DB2 transforms the full join into a left join, which is more efficient:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2 > 12;
```

Predicates that eliminate null values

In the following statement, the `X.C2>12` predicate filters out all null values that result from the right join:

```
SELECT * FROM T1 X RIGHT JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2>12;
```

Therefore, DB2 can transform the right join into a more efficient inner join without changing the result:

```
SELECT * FROM T1 X INNER JOIN T2 Y
  ON X.C1=Y.C1
  WHERE X.C2>12;
```

Predicates that follow join operations

The predicate that follows a join operation must have the following characteristics before DB2 transforms an outer join into a simpler outer join or inner join:

- The predicate is a Boolean term predicate.
- The predicate is false if one table in the join operation supplies a null value for all of its columns.

The following predicates are examples of predicates that can cause DB2 to simplify join operations:

```
T1.C1 > 10
T1.C1 IS NOT NULL
T1.C1 > 10 OR T1.C2 > 15
T1.C1 > T2.C1
T1.C1 IN (1,2,4)
T1.C1 LIKE 'ABC%'
T1.C1 BETWEEN 10 AND 100
12 BETWEEN T1.C1 AND 100
```

ON clauses that eliminate unmatched values

This examples shows how DB2 can simplify a join operation because the query contains an ON clause that eliminates rows with unmatched values:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  FULL JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

Because the last ON clause eliminates any rows from the result table for which column values that come from T1 or T2 are null, DB2 can replace the full join with a more efficient left join to achieve the same result:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  LEFT JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

Full outer joins processed as left outer joins

In one case, DB2 transforms a full outer join into a left join when you cannot write code to do it. This is the case where a view specifies a full outer join, but a subsequent query on that view requires only a left outer join.

For example, consider the view that is created by the following statement:

```
CREATE VIEW V1 (C1,T1C2,T2C2) AS
  SELECT COALESCE(T1.C1, T2.C1), T1.C2, T2.C2
  FROM T1 X FULL JOIN T2 Y
  ON T1.C1=T2.C1;
```

This view contains rows for which values of C2 that come from T1 are null. However, if you execute the following query, you eliminate the rows with null values for C2 that come from T1:

```
SELECT * FROM V1
  WHERE T1C2 > 10;
```

Therefore, for this query, a left join between T1 and T2 would have been adequate. DB2 can execute this query as if the view V1 was generated with a left outer join so that the query runs more efficiently.

Removal of unneeded tables in left outer joins

When an SQL statement contains a left outer join, but does not select any columns from the right side of the join, DB2 can remove the join from the statement.

The right table is unneeded if either of the following conditions are true:

- A unique index exists on the join key column of the right table.
- The statement specifies SELECT DISTINCT.

For example consider the following statement:

```
SELECT DISTINCT T1.C3
FROM T1 LEFT OUTER JOIN T2
ON T1.C2 = T2.C2
WHERE T1.C1 = ?
```

Because the statement specified SELECT DISTINCT, the reference to the right table is unneeded, and DB2 can select an access path for the following statement instead:

```
SELECT DISTINCT T1.C3
FROM T1
WHERE T1.C1 = ?
```

Because all references to the right table have been removed from the statement, the PLAN_TABLE output and access path diagrams for the statement contain no references to the table.

PSPI

Related concepts:

Query transformations

Related reference:

PLAN_TABLE

Removal of pre-evaluated predicates

In certain situations DB2 might modify SQL statements to remove unneeded predicates that can always be pre-evaluated. The removal of these predicates might facilitate selection of more efficient access paths.

PSPI

Always-false predicates

DB2 sometimes removes predicates that are always evaluated as false. Such predicates are sometimes used by query generators, and in application programs, to toggle between “real” and “fake” predicates. In many cases DB2 can remove these predicates to improve the efficiency of the access path for the statement.

DB2 might remove certain always-false predicates, as described by the following examples:

- Equal predicates that contain non-matching constant values: `constant-value1 = constant-value2`
- IN predicates that match a constant value to a list of non-matching constant values: `constant-value1 IN (constant-value2, constant-value3,constant-value3)`

- IS NULL predicates for a column that is defined as NOT NULL: *non-null-col* IS NULL

When DB2 encounters always false predicates in complex predicates that contain OR and AND conditions, it uses certain rules to determine whether to remove always-false predicates. For example:

- Always-false predicates under an OR condition might be removed. For example, consider the following predicate:

  ```
  WHERE ('A' = 'B' OR COL1 IN ('B', 'C'))
  ```

  DB2 can simplify the predicate by removing the `'A' = 'B'` predicate, which is always false. Therefore DB2 can use the following simplified predicate instead:

  ```
  WHERE COL1 IN ('B', 'C')
  ```

- Always-false predicates that compare non-matching constant values under an AND condition are not removed. For example, DB2 does not simplify the following predicates:

  ```
    C1 = 1 AND 1 = 3
    C1 = 1 OR 1 = 2 AND 3 = 4
    C1 = 1 OR 1 = 2 AND 3 = 4
  ```

- DB2 does not remove `OR 0=1` or `OR 0<>0` predicates. Predicates of these forms can be used to prevent index access.
- IS NULL predicates (for NOT NULL columns) under AND conditions are removed. If the AND condition is under an OR condition, the OR condition is also removed. For example, consider the following statement:

  ```
  SELECT * FROM T1
  WHERE (C1 IS NULL AND C2 > 123) OR C3 = 54321;
  ```

  DB2 can simplify this statement to the following form:

  ```
  SELECT * FROM T1 WHERE C3 = 54321;
  ```

- When an always-false IS NULL predicate is under an OR condition and the other side contains a host variable or expression, the predicate is not removed.
- Predicates are removed from only WHERE, HAVING, and ON clauses.
- WHEN predicates for CASE expressions are not simplified.
- Always-false range predicates that compare constants, such as `1 > 2` are not removed.
- Predicates that contains host variables, which might be always-false under REOPT(ALWAYS), are not removed.
- Constant range predicates, such as `1 > 2` are not pre-evaluated as always-false predicates
- Predicates with host variables such as `:H1='A'` are not pre-evaluated, even under REOPT(VARS) processing rules.
- If one side of an AND or OR predicate is a predicate that involves a subquery, the always-false result cannot be propagated through it.

Related concepts:

How DB2 simplifies join operations

Using EXPLAIN to determine UNION, INTERSECT, and EXCEPT activity and query rewrite

Predicates that DB2 generates

DB2 might generate certain predicates to improve access path selection.

PSPI

The goal of most generated predicates is to increase the likelihood that DB2 can select more efficient and index-based access paths for a query. DB2 generates these predicates automatically, and you generally do not need to take action to benefit from them. However, you might notice the additional predicates in EXPLAIN output, and when you use query tuning tools to view access path diagrams or formatted query text.

DB2 might remove either the generated predicate or the original predicate if it is not required for the selected access path.

DB2 might generate additional predicates when it encounters the following types of situations:

Predicate through transitive closure

DB2 might generate additional predicates through transitive closure to improve access path efficiency. Transitive closure occurs when several predicates taken in combination logically imply the existence of additional predicates. Because those implied predicates might enable the selection of a more efficient access path, DB2 might generate such predicates.

PSPI

Related concepts:

Predicates generated through transitive closure

Query transformations

Stage 1 and stage 2 predicates

Indexable and non-indexable predicates

Interpreting data access by using EXPLAIN

Related tasks:

Generating visual representations of access plans (IBM Data Studio)

Formatting SQL statements (IBM Data Studio)

Related reference:

DSN_PREDICAT_TABLE

Predicates generated through transitive closure

When the set of predicates that belong to a query logically imply other predicates, DB2 can generate additional predicates to provide more information for access path selection.

A basic example of transitive closure is a query that contains both of the following predicates: `COL1=COL2` and `COL2=COL3`. Given these two predicates, logic implies that a third predicate `COL1=COL3` is also valid, even though that predicate might not exist in the statement. In the event that this third predicate might enable DB2 to choose a more efficient access path, DB2 might generate the additional predicate.

Rules for generating predicates

PSPI

For single-table or inner join queries, DB2 might generate predicates for transitive if the following conditions are true:

- The query has an equal type local or join predicate, such as: `COL1=COL2`. The query also has a Boolean term predicate on one of the columns in the first predicate, with one of the following formats:
- The query also contains a Boolean term predicate on one of the columns in the first predicate, with one of the following formats:
  - `COL1` *`op value`*

    op is =, <>, >, >=, <, or <=.

    value is a constant, host variable, or special register.
  - `COL1 (NOT) BETWEEN` *`value1`* `AND` *`value2`*
  - `COL1=COL3`
- The query contains an IN-list predicate, such as `COL1 IN (`*`value1`*`,` *`value2`*`,` *`value3`*`)`
- The query is an outer join query and has an ON clause in the form of `COL1=COL2` that comes before a join that has one of the following forms:
  - `COL1` *`op value`*

    op is =, , >, >=, <, or <=
  - `COL1 (NOT) BETWEEN` *`value1`* `AND` *`value2`*

DB2 generates a transitive closure predicate for an outer join query only if the generated predicate does not reference the table with unmatched rows. In other words, the generated predicate cannot reference the left table for a left outer join or the right table for a right outer join.

For a multiple-CCSID query, DB2 does not generate a transitive closure predicate if the predicate that would be generated has any of the following characteristics:

- The generated predicate is a range predicate (*op* is >, >=, <, or <=).
- Evaluation of the query with the generated predicate results in different CCSID conversion from evaluation of the query without the predicate.

When a predicate meets the transitive closure conditions, DB2 generates a new predicate, whether or not it already exists in the WHERE clause.

The generated predicates have one of the following formats:

- `COL` *`op value`*

  op is =, >, >=, <, or <=.

  value is a constant, host variable, or special register.
- `COL (NOT) BETWEEN` *`value1`* `AND` *`value2`*
- `COL1=COL2` (for single-table or inner join queries only)

DB2 does not generate a predicate through transitive closure for any predicate that is associated with the DECFLOAT data type (column or constant).

Example of transitive closure for an inner join: Suppose that you have written this query, which meets the conditions for transitive closure:

```
SELECT * FROM T1, T2
  WHERE T1.C1=T2.C1 AND
  T1.C1>10;
```

DB2 generates an additional predicate to produce this query, which is more efficient:

```
SELECT * FROM T1, T2
  WHERE T1.C1=T2.C1 AND
  T1.C1>10 AND
  T2.C1>10;
```

Example of transitive closure for an outer join

Suppose that you have written this outer join query:

```
SELECT * FROM
  (SELECT T1.C1 FROM T1 WHERE T1.C1>10) X
  LEFT JOIN
  (SELECT T2.C1 FROM T2)                Y
  ON X.C1 = Y.C1;
```

The before join predicate, T1.C1>10, meets the conditions for transitive closure, so DB2 generates a query that has the same result as this more-efficient query:

```
SELECT * FROM
  (SELECT T1.C1 FROM T1 WHERE T1.C1>10) X
  LEFT JOIN
  (SELECT T2.C1 FROM T2 WHERE T2.C1>10) Y
  ON X.C1 = Y.C1;
```

Example of transitive closure for an IN-list predicate

Assume that the following objects exist:

- Two tables are defined by the following statements:

  ```
  CREATE TABLE CAMP
    (NAME CHAR(128),
    THEME CHAR(64) ,
    LOCATION CHAR(64),
    ...)
  ```

  ```
  CREATE TABLE STUDENT
    (NAME CHAR(128),
    THEME CHAR(64) ,
    LOCATION CHAR(64),
    ...)
  ```

- Two indexes are defined by the following statements:

  ```
  Index IX_CAMP on CAMP(THEME, LOCATION)
  ```

  ```
  Index IX_STUDENT on STUDENT(THEME, LOCATION)
  ```

Consider the following query:

```
SELECT C.NAME 'Camp Name', S.NAME 'Student Name'
FROM
CAMP C,
STUDENT S
WHERE
C.THEME = S.THEME
AND C.LOCATION = S.LOCATION
AND S.THEME IN ('theatre', 'jazz')
AND S.LOCATION IN ('monterey', 'carmel')
```

Through transitive closure, DB2 can generate two predicates to enable more access path options:

```
SELECT C.NAME 'Camp Name', S.NAME 'Student Name'
FROM
CAMP C,
STUDENT S
WHERE
C.THEME = S.THEME
AND C.LOCATION = S.LOCATION
AND S.THEME IN ('theatre', 'jazz')
AND S.LOCATION IN ('monterey', 'carmel')
AND C.THEME IN ('theatre', 'jazz')
AND C.LOCATION IN ('monterey', 'carmel')
```

When generating transitive closure for IN list predicates, DB2 sorts the IN-list elements and removes duplicates if the list contains only constants.

Predicate redundancy for transitive closure

A predicate is redundant if evaluation of other predicates in the query already determines the result that the predicate provides. You can specify redundant predicates for transitive closure, or DB2 can generate them. However, DB2 does not determine that any of your query predicates are redundant. Therefore, all predicates that you code are evaluated at execution time, regardless of whether they are logically redundant. In contrast, if DB2 generates a redundant predicate for improved access path selection, DB2 can ignore that predicate at execution.

PSPI

Related concepts:

Query transformations

How DB2 modifies IN predicates

Related tasks:

Adding extra predicates to improve access paths

Transformation of SQL predicates to XML predicates

DB2 sometimes transforms an SQL query to change the timing at which a predicate is applied to improve the performance of the query. DB2 might use such a transformation to push SQL predicates into the XPath expression embedded in the XMLTABLE function.

PSPI

For example, a query finds all books that were published after 1991 and lists the year, title and publisher for each book.

```
SELECT X.*
FROM T1,
 XMLTABLE('/bib/book'
      PASSING T1.bib_xml
      COLUMNS YEAR INT PATH '@year',
       TITLE VARCHAR(30) PATH 'title',
      PUBLISHER VARCHAR(30) PATH 'publisher') X
WHERE X.YEAR > 1991;
```

DB2 can rewrite the query to process the `WHERE X.YEAR > 1991` predicate in the XMLTABLE function. In the rewritten query the original predicate becomes an XPath predicate that is associated with the *row-xpath-expression* of the XMLTABLE function:

```
SELECT X.*
FROM T1,
 XMLTABLE('/bib/book[@year>1991]'
     PASSING T1.bib_xml
     COLUMNS YEAR INT PATH '@year',
        TITLE VARCHAR(30) PATH 'title',
      PUBLISHER VARCHAR(30) PATH 'publisher') X
```

Implications of truncation and trailing blanks

Unlike SQL, in which trailing blanks have no significance, in XPath trailing blanks are significant. For example, the following query contains an additional predicate, `X.publisher = 'Addison-Wesley'`:

```
SELECT *
FROM T1,
 XMLTABLE('/bib/book'
     PASSING T1.bib_xml
     COLUMNS year INT PATH '@year',
        title VARCHAR(30) PATH 'title',
       publisher VARCHAR(30) PATH 'publisher') X
WHERE X.year > 1991
 AND X.publisher = 'Addison-Wesley';
```

Because of the possible truncation when a publisher is cast to varchar(30), and the possibility of trailing blanks in the original XML data, DB2 must add an internal operator, db2:rtrim, to simulate the SQL semantics in order to push the predicate into XPath. As shown below. The predicate `X.publisher = 'Addison-Wesley'` is transformed into `[db2:rtrim(publisher,30)="Addison-Wesley"]`.

Predicates that are eligible for transformation to XML predicates in XMLTABLE

A predicate that satisfies the following criteria is eligible for transformation to be processed by the XMLTABLE function:

- The predicate must have one of the following forms: (Where *op* stands for any of the following operators: =, <, >, <=. >=, or <>.)
  - Column *op* constant, parameter, or host variable, where the column is from the result table.
  - Column *op* column, where the column on the left hand side is from the result table and the column and the right hand side is from either the result table or one of the input tables.
  - Column *op* expression, where the column is from the result table and the expression is any SQL expression that only contains columns from the input table.
  - A BETWEEN predicate that can be transformed into one of the above forms.
  - COLUMN IS (NOT) NULL
  - A predicate that is composed of the above forms combined with AND and OR.
  - COLUMN (NOT) IN (expression 1, ..., expression *n*), where the column is from the result table and each of the expressions on either a column from the result table or an SQL expression that contains neither columns from the result table nor columns from a table that is NOT an input table.
- The predicate is a boolean term predicate.
- The predicate can be applied before any join operations.

- The result column of the XMLTABLE function that is involved in the predicate is not of any of the following data types:

  DATE
  TIME
  TIMESTAMP
  DECFLOAT(16)
  REAL
  DOUBLE

  This restriction does not apply to IS (NOT) NULL predicate.
- The result column of the XMLTABLE function involved in the predicate does not have a default clause.
- The XMLTABLE function does not have a FOR ORDINALITY column.

PSPI

Related concepts:

Query transformations

XMLTABLE function for returning XQuery results as a table (DB2 Programming for XML)

Related reference:

XMLTABLE (DB2 SQL)

Predicates with encrypted data

DB2 provides built-in functions for data encryption and decryption. These functions can secure sensitive data, but they can also degrade the performance of some statements if they are not used carefully.

PSPI

If a predicate contains any operator other than = and <>, encrypted data must be decrypted before comparisons can be made. Decryption makes the predicates stage 2.

PSPI

Making predicates eligible for expression-based indexes

You can create an expression-based index to improve the performance of queries that use column-expression predicates.

About this task

Introductory concepts

Types of indexes (Introduction to DB2 for z/OS)
Expressions (DB2 SQL)
Index keys (Introduction to DB2 for z/OS)

PSPI

Unlike a simple indexes, an expression-based index uses key values that are transformed by an expression that is specified when the index is created. However, DB2 cannot always use an expression-based index. For example, DB2 might not be able to use an index on expression for queries that contain multiple outer joins, materialized views, and materialized table expressions.

DB2 does not use an expression-based index for queries that use sensitive static cursors.

Procedure

To enable DB2 to use an expression-based index, use the following approaches:

- Create an expression-based index for queries that contain predicates that use column-expressions.
- Rewrite queries that contain multiple outer joins so that a predicate that can be satisfied by an index on expression is in a different query block than the outer joins. For example, DB2 cannot use an expression-based index for the UPPER predicate in the following query:

  ```
  SELECT ...
  FROM T1
  LEFT OUTER JOIN T2
    ON T1.C1 = T2.C1
  LEFT OUTER JOIN T3
    ON T1.C1 = T3.C1
  WHERE UPPER(T1.C2, 'EN_US') = 'ABCDE'
  ```

  However, you can rewrite the query so that DB2 can use an expression-based index for the UPPER predicate by placing the UPPER expression in a separate query block from the outer joins, as shown in the following query:

  ```
  SELECT ...
  FROM (
    SELECT C1
    FROM T1
    WHERE UPPER(T1.C2, 'EN_US') = 'ABCDE'
  ) T1
  LEFT OUTER JOIN T2
    ON T1.C1 = T2.C1
  LEFT OUTER JOIN T3
    ON T1.C1 = T3.C1
  ```

- Rewrite queries that contain materialized views and table expressions so that any predicate that might benefit from an expression-based index is coded inside the view or table expression. For example, in the following query as written, the table expression X is materialized because of the DISTINCT keyword, and DB2 cannot use an expression-based index for the UPPER predicate:

  ```
  SELECT ...
  FROM (
    SELECT DISTINCT C1, C2 FROM T1
  ) X
  , T2
  WHERE X.C1 = T2.C1
    AND UPPER(X.C2, 'En_US') = 'ABCDE'
  ```

  However, you can enable DB2 to use an expression-based index for the UPPER predicate by rewriting the query so that the UPPER predicate is coded inside the table expression, as shown in the following example:

  ```
  SELECT ...
  FROM (
    SELECT DISTINCT C1, C2 FROM T1
  ```

```
  WHERE UPPER(T1.C2, 'En_US') = 'ABCDE'
) X
, T2
WHERE X.C1 = T2.C1
```

PSPI

Related concepts:

Query transformations

Expression-based indexes (Introduction to DB2 for z/OS)

Capability to create an expression-based index (DB2 for z/OS What's New?)

Using host variables efficiently

When host variables or parameter markers are used in a query, the actual values are not known when you bind the package or plan that contains the query. DB2 uses a default filter factor to determine the best access path for an SQL statement. If that access path proves to be inefficient, you can do several things to obtain a better access path.

About this task

PSPI

Host variables require default filter factors. When you bind a static SQL statement that contains host variables, DB2 uses a default filter factor to determine the best access path for the SQL statement. DB2 often chooses an access path that performs well for a query with several host variables. However, in a new release or after maintenance has been applied, DB2 might choose a new access path that does not perform as well as the old access path. In many cases, the change in access paths is due to the default filter factors, which might lead DB2 to optimize the query in a different way.

Procedure

To change the access path for a query that contains host variables, use one of the following actions:

- Bind the package or plan that contains the query with the options REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE).
- Rewrite the query.

PSPI

Related tasks:

Reoptimizing SQL statements at run time

Writing efficient subqueries

A *subquery* is a SELECT statement within the WHERE or HAVING clause of an INSERT, UPDATE, MERGE, or DELETE SQL statement. By understanding how DB2 processes subqueries, you can estimate the best method to use when writing a given query when several methods can achieve the same result.

About this task

PSPI

In many cases two or more different SQL statements can achieve identical results, particularly those that contain subqueries. The statements have different access paths, however, and probably perform differently.

Subqueries might also contain their own subqueries. Such *nested subqueries* can be either correlated or non-correlated. DB2 uses the same processing techniques with nested subqueries that it does for non-nested subqueries, and the same optimization techniques apply.

No absolute rules exist for deciding how or whether to code a subquery. DB2 might transform one type of subquery to another, depending on the optimizer estimation.

Procedure

To ensure the best performance from SQL statements that contain subqueries:

Follow these general guidelines:

- If efficient indexes are available on the tables in the subquery, then a correlated subquery is likely to be the most efficient kind of subquery.
- If no efficient indexes are available on the tables in the subquery, then a non-correlated subquery would be likely to perform better.
- If multiple subqueries are in any parent query, make sure that the subqueries are ordered in the most efficient manner.

Example

Assume that MAIN_TABLE has 1000 rows:

```
SELECT * FROM MAIN_TABLE
  WHERE TYPE IN (subquery 1) AND
        PARTS IN (subquery 2);
```

Assuming that subquery 1 and subquery 2 are the same type of subquery (either correlated or non-correlated) and the subqueries are stage 2, DB2 evaluates the subquery predicates in the order they appear in the WHERE clause. Subquery 1 rejects 10% of the total rows, and subquery 2 rejects 80% of the total rows:

- The predicate in subquery 1 (which is referred to as P1) is evaluated 1000 times, and the predicate in subquery 2 (which is referred to as P2) is evaluated 900 times, for a total of 1900 predicate checks. However, if the order of the subquery predicates is reversed, P2 is evaluated 1000 times, but P1 is evaluated only 200 times, for a total of 1200 predicate checks.
- Coding P2 before P1 appears to be more efficient if P1 and P2 take an equal amount of time to execute. However, if P1 is 100 times faster to evaluate than P2, then coding subquery 1 first might be advisable. If you notice a performance degradation, consider reordering the subqueries and monitoring the results.

  If you are unsure, run EXPLAIN on the query with both a correlated and a non-correlated subquery. By examining the EXPLAIN output and understanding your data distribution and SQL statements, you should be able to determine which form is more efficient.

This general principle can apply to all types of predicates. However, because subquery predicates can potentially be thousands of times more processor- and I/O-intensive than all other predicates, the order of subquery predicates is particularly important.

Regardless of coding order, DB2 performs non-correlated subquery predicates before correlated subquery predicates, unless the subquery is transformed into a join.

PSPI

Correlated and non-correlated subqueries

Different subqueries require different approaches for efficient processing by DB2.

All subqueries can be classified into either two categories: correlated and non-correlated

Correlated subqueries

Correlated subqueries contain a reference to a table or column that is outside of the scope of the subquery.

GUPI

In the following query, for example, the correlation name X is a value from a table that is not listed in the FROM clause of the subquery. The inclusion of X illustrates that the subquery references the outer query block:

```
SELECT * FROM DSN8A10.EMP X
   WHERE  JOB = 'DESIGNER'
      AND  EXISTS (SELECT 1
                  FROM   DSN8A10.PROJ
                  WHERE  DEPTNO = X.WORKDEPT
                     AND  MAJPROJ = 'MA2100');
```

GUPI

Non-correlated subqueries

Non-correlated subqueries do not refer to any tables or columns that are outside of the scope of the subquery.

GUPI

The following example query refers only to tables are within the scope of the FROM clause.

```
SELECT * FROM DSN8A10.EMP
  WHERE  JOB = 'DESIGNER'
    AND  WORKDEPT IN (SELECT DEPTNO
                      FROM   DSN8A10.PROJ
                      WHERE  MAJPROJ = 'MA2100');
```

GUPI

Related concepts:

Correlated subqueries (DB2 Application programming and SQL)

When DB2 transforms a subquery into a join

DB2 might sometimes transform a subquery into a join for SELECT, UPDATE, and DELETE statements.

PSPI

When SELECT, UPDATE and DELETE statements contain subqueries, DB2 might transform the statements to use joins instead of the subqueries. The order of access for a subquery is more restrictive than for an equivalent join. Therefore, DB2 might gain more flexibility for optimizing an access path by using the join.

However, DB2 cannot always transform every subquery to a join. For example, DB2 does not transform a query when the transformation would introduce redundancy, or when the subquery contains certain clauses or predicate types.

However, the specific criteria for transformation are not described here. The purpose of the transformation is to provide the additional flexibility for optimization without a rewrite. Consequently, the recommendation for any rewrite is to use the join explicitly rather than making the subquery eligible for transformation.

The following subquery can be transformed into a join because it meets the above conditions for transforming a SELECT statement:

```
SELECT * FROM EMP
  WHERE DEPTNO IN
   (SELECT DEPTNO FROM DEPT
      WHERE LOCATION IN ('SAN JOSE', 'SAN FRANCISCO')
      AND DIVISION = 'MARKETING');
```

If a department in the marketing division has branches in both San Jose and San Francisco, the result of the SQL statement is not the same as if a join were performed. The join makes each employee in this department appear twice because it matches once for the department of location San Jose and again of location San Francisco, although it is the same department. Therefore, it is clear that to transform a subquery into a join, the uniqueness of the subquery select list must be guaranteed. For this example, a unique index on any of the following sets of columns would guarantee uniqueness:

- (DEPTNO)
- (DIVISION, DEPTNO)
- (DEPTNO, DIVISION).

The resulting query after the transformation has the following form:

```
SELECT EMP.* FROM EMP, DEPT
   WHERE EMP.DEPTNO = DEPT.DEPTNO AND
         DEPT.LOCATION IN ('SAN JOSE', 'SAN FRANCISCO') AND
         DEPT.DIVISION = 'MARKETING';
```

PSPI

When DB2 correlates and de-correlates subqueries

Correlated and non-correlated subqueries have different processing advantages. DB2 transforms subqueries to whichever type is most efficient, especially when a subquery cannot be transformed into a join.

PSPI

DB2 might transform a correlated query to a non-correlated, or *de-correlate* the subquery, to improve processing efficiency. Likewise, DB2 the might *correlate* a non-correlated subquery. When a correlated and non-correlated subquery can achieve the same result, the most efficient way depends on the data.

DB2 chooses to correlate or de-correlate subqueries based on cost. Correlated subqueries allow more filtering to be done within the subquery. Non-correlated subqueries allow more filtering to be done on the table whose columns are being compared to the subquery result.

DB2 might correlate a non-correlated subquery, or de-correlate a correlated subquery, that cannot be transformed into a join to improve access path selection and processing efficiency.

Example:

DB2 can transform the following non-correlated subquery into a correlated subquery:

```
SELECT * FROM T1
  WHERE T1.C1 IN (SELECT T2.C1 FROM T2, T3
                  WHERE T2.C1 = T3.C1)
```

Can be transformed to:

```
SELECT * FROM T1
  WHERE EXISTS (SELECT 1 FROM T2, T3
                WHERE T2.C1 = T3.C1 AND T2.C1 = T1.C1)
```

Some queries cannot be transformed from one form to another. Most set functions and grouping functions make it difficult to transform a subquery from one form to another. Expressions that can prevent such transformation include:

Set functions and grouping functions

Most set functions and grouping functions make it difficult to transform a subquery from one form to another.

Example:

In the following query, the non-correlated subquery cannot be correlated to T1 because it would change the result of the SUM function. Consequently, only the non-correlated form of the query can be considered.

```
SELECT * FROM T1
WHERE T1.C2 IN (SELECT SUM(T2.C2) FROM T2, T3
       WHERE T2.C1 = T3.C1
       GROUP BY T2.C1)
```

Correlated ranges and <> comparisons

Some range comparisons involving correlated columns make it difficult to de-correlate the subquery. This is because when a correlated subquery is de-correlated we might have to remove duplicates in order to consider the virtual table in the outer position (see “Early Out” Processing). This duplicate removal requires a set of “equal-join” predicate columns as the key. Without equal-join predicates the early out process breaks down and doesn't work. That means the virtual table can only be considered in correlated form (as the inner table of the join).

Example:

DB2 cannot de-correlate the following query and use it to access T1 because removing duplicates on the T2.C2 subquery result does not guarantee that the range predicate correlation does not qualify multiple rows from T1.

```
SELECT * FROM T1
  WHERE EXISTS (SELECT 1 FROM T2, T3
                WHERE T2.C1 = T3.C1 AND T2.C2 > T1.C2 AND T2.C2 < T1.C3)
```

PSPI

Subquery tuning

DB2 automatically performs some subquery tuning by subquery to join transformation and through subquery correlation and de-correlation.

PSPI

However, you should be aware of the differences among the subqueries such as those in the following examples. You might need to code a query in one of the ways below for performance reasons that stem from restrictions to DB2 in transforming a given query, or because DB2 cannot accurately estimate the cost of the various transformation choices during query optimization.

Each of the following three queries retrieves the same rows. All three retrieve data about all designers in departments that are responsible for projects that are part of major project MA2100. These three queries show that you can retrieve a result in several ways.

Query A: a join of two tables

```
SELECT DSN8A10.EMP.* FROM DSN8A10.EMP, DSN8A10.PROJ
    WHERE  JOB = 'DESIGNER'
   AND  WORKDEPT = DEPTNO
  AND  MAJPROJ = 'MA2100';
```

Query B: a correlated subquery

```
SELECT * FROM DSN8A10.EMP X
   WHERE  JOB = 'DESIGNER'
     AND  EXISTS (SELECT 1 FROM DSN8A10.PROJ
                  WHERE  DEPTNO = X.WORKDEPT
                    AND  MAJPROJ = 'MA2100');
```

Query C: a noncorrelated subquery

```
SELECT * FROM DSN8A10.EMP
    WHERE  JOB = 'DESIGNER'
      AND  WORKDEPT IN (SELECT DEPTNO FROM DSN8A10.PROJ
                        WHERE  MAJPROJ = 'MA2100');
```

Choosing between a subquery and a join

If you need columns from both tables EMP and PROJ in the output, you must use the join. Query A might be the one that performs best, and as a general practice you should code a subquery as a join whenever possible. However, in this example, PROJ might contain duplicate values of DEPTNO in the subquery, so that an equivalent join cannot be written. In that case, whether the correlated or non-correlated form is most efficient depends upon where the application of each predicate in the subquery provides the most benefit.

When looking at a problematic subquery, check if the query can be rewritten into another format, especially as a join, or if you can create an index to improve the performance of the subquery. Consider the sequence of evaluation for the different subquery predicates and for all other predicates in the query. If one subquery predicate is costly, look for another predicate that could be evaluated first to reject more rows before the evaluation of problem subquery predicate.

PSPI

Related concepts:

Correlated subqueries (DB2 Application programming and SQL)

When DB2 transforms a subquery into a join

Investigating join operations

Related tasks:

Using predicates efficiently

Joining data from more than one table (DB2 Application programming and SQL)

Query transformations

DB2 sometimes modifies the text of SQL statements to improve access path efficiency.

Query transformations become most important for complex queries, especially complex queries that are created by query generators. DB2 might apply the following types of transformations to SQL statements, among others:

- Removal of unneeded or pre-evaluated predicates
- Addition of generated predicates
- Removal of table references for certain joins
- Correlation or de-correlation of subqueries
- Conversion of subqueries to joins
- Simplification and removal of subselects in statements that contain UNION ALL operators

Because of the complexity of such queries, exhaustive description of every case for transformation and every restriction is not practical. Therefore, the transformations are described at only a very a high level with basic examples. None of the transformations are guaranteed to occur for any particular statement.

In most cases, you do not have to do anything to take advantage of such transformations. However, you might notice some of these changes in

PLAN_TABLE output. You might also see such changes when you use query tuning tools, such as IBM Data Studio, to format SQL statement text and view access path diagrams.

The recommendation to write queries as simply as possible always remains. There, you would not, in most cases, try to rewrite a simpler query into a more complex form for the purpose of make it eligible for a particular transformation.

Related concepts:

Predicate manipulation

When DB2 correlates and de-correlates subqueries

When DB2 transforms a subquery into a join

Using EXPLAIN to determine UNION, INTERSECT, and EXCEPT activity and query rewrite

Related tasks:

Generating visual representations of access plans (IBM Data Studio)

Formatting SQL statements (IBM Data Studio)

Related reference:

PLAN_TABLE

Materialized query tables and query performance

One way to improve the performance of dynamic queries that operate on very large amounts of data is to generate the results of all or parts of the queries in advance, and store the results in materialized query tables.

PSPI

Materialized query tables are user-created tables. Depending on how the tables are defined, they are user-maintained or system-maintained. If you have set subsystem parameters or an application sets special registers to tell DB2 to use materialized query tables, when DB2 executes a dynamic query, DB2 uses the contents of applicable materialized query tables if DB2 finds a performance advantage to doing so.

PSPI

Encrypted data and query performance

Encryption and decryption can degrade the performance of some queries.

However, you can lessen the performance impact of encryption and decryption by writing your queries carefully and designing your database with encrypted data in mind.

XML data and query performance

XML data is more expensive to process, and might affect the performance of most SQL statements. Each row of an XML column contains an XML document, requires more processing, and requires more space in DB2.

PSPI

When you use an XQuery or XPath expression to search or extract the XML data, you can lessen the performance impact by avoiding the descendant or descendant-or-self axis in the expression. The XMLEXISTS predicate is always stage 2. However, you can use the XML index to reduce the number of rows, that is, the number of XML documents, that must be searched at the second stage.

Creating and maintaining XML indexes might be more costly than non-XML indexes. Write queries to use non-XML indexes to filter as many rows as possible before the second stage.

PSPI

Related concepts:

Stage 1 and stage 2 predicates

Best practices for XML performance in DB2

Best applications for XQuery or XPath (DB2 Programming for XML)

XML data indexing (DB2 Programming for XML)

Filter factor estimation for the XMLEXISTS predicate

Examples of XML index usage by predicates that test for node existence (DB2 Programming for XML)

Access methods with XML indexes (DB2 Programming for XML)

XML index attributes (Introduction to DB2 for z/OS)

Related tasks:

Matching index scan (MATCHCOLS>0)

Related reference:

XMLEXISTS predicate (DB2 SQL)

Using scrollable cursors efficiently

Scrollable cursors are a valuable tool for writing applications such as screen-based applications, in which the result table is small and you often move back and forth through the data.

Procedure

PSPI

To get the best performance from your scrollable cursors:

- Determine when scrollable cursors work best for you.

  Scrollable cursors require more DB2 processing than non-scrollable cursors. If your applications require large result tables or you only need to move sequentially forward through the data, use non-scrollable cursors.
- Declare scrollable cursors as SENSITIVE only if you need to see the latest data.

  If you do not need to see updates that are made by other cursors or application processes, using a cursor that you declare as INSENSITIVE requires less processing by DB2.

  If you need to see only some of the latest updates, and you do not need to see the results of insert operations, declare scrollable cursors as SENSITIVE STATIC.

If you need to see all of the latest updates and inserts, declare scrollable cursors as SENSITIVE DYNAMIC.

- To ensure maximum concurrency when you use a scrollable cursor for positioned update and delete operations, specify ISOLATION(CS) and CURRENTDATA(NO) when you bind packages and plans that contain updatable scrollable cursors.
- Use the FETCH FIRST *n* ROWS ONLY clause with scrollable cursors when it is appropriate. In a distributed environment, when you need to retrieve a limited number of rows, FETCH FIRST *n* ROWS ONLY can improve your performance for distributed queries that use DRDA access by eliminating unneeded network traffic.

  In a local environment, if you need to scroll through a limited subset of rows in a table, you can use FETCH FIRST *n* ROWS ONLY to make the result table smaller.
- In a distributed environment, if you do not need to use your scrollable cursors to modify data, do your cursor processing in a stored procedure. Using stored procedures can decrease the amount of network traffic that your application requires.
- In a work file database, create table spaces that are large enough for processing your scrollable cursors.

  DB2 uses declared temporary tables for processing the following types of scrollable cursors:
  - SENSITIVE STATIC SCROLL
  - INSENSITIVE SCROLL
  - ASENSITIVE SCROLL, if the cursor sensitivity in INSENSITIVE. A cursor that meets the criteria for a read-only cursor has an effective sensitivity of INSENSITIVE.
- Commit changes often for the following reasons:
  - You frequently need to leave scrollable cursors open longer than non-scrollable cursors.
  - An increased chance of deadlocks with scrollable cursors occurs because scrollable cursors allow rows to be accessed and updated in any order. Frequent commits can decrease the chances of deadlocks.
  - To prevent cursors from closing after commit operations, declare your scrollable cursors WITH HOLD.
- Use the following methods to prevent false out-of-space indications:
  1. Check applications such that they commit frequently.
  2. Close sensitive scrollable cursors that as soon as they are no longer needed.
  3. Remove WITH HOLD option for the sensitive scrollable cursor, if possible.
  4. Isolate LOB table spaces in a dedicated buffer pool in the data sharing environment.

  While sensitive static scrollable cursors are open against a table, DB2 disallows reuse of space in that table space to prevent the scrollable cursor from fetching newly inserted rows that were not in the original result set. Although this is normal, it can result in a seemingly false out-of-space indication. The problem can be more noticeable in a data sharing environment with transactions that access LOBs.

  In addition to the space reuse issue, the use of a sensitive static scrollable cursor in a data sharing environment might also result in lock contention on INSERT statements if the inserted objects are in the same buffer pool. This situation applies regardless of whether the objects have sensitive static scrollable cursors, and regardless of whether the objects contain any LOB columns. You can minimize this problem by isolating objects that have a large volume of insert activity so that they are in a dedicated buffer pool within the data sharing environment.

- Do not specify a sensitive static scrollable cursor when index access is needed for an expression-based index.

PSPI

Related concepts:

Types of cursors (DB2 Application programming and SQL)

Related tasks:

Retrieving rows by using a scrollable cursor (DB2 Application programming and SQL)

Efficient queries for tables with data-partitioned secondary indexes

The number of partitions that DB2 accesses to evaluate a query predicate can affect the performance of the query. A query that provides data retrieval through a data-partitioned secondary index (DPSI) might access some or all partitions of the DPSI.

Introductory concepts

Indexes on partitioned tables (Introduction to DB2 for z/OS)

PSPI

For a query that is based only on a DPSI key value or range, DB2 must examine all partitions. However, if the query also has predicates on the leading columns of the partitioning key, DB2 does not need to examine all partitions. The removal from consideration of inapplicable partitions is known as *page range screening*, which is also sometimes known called *limited partition scan*.

The use of page range screening scan can be determined at bind time or at run time. For example, page range screening can be determined at bind time for a predicate in which a column is compared to a constant. Page range screening occurs at run time if the column is compared to a host variable, parameter marker, or special register.

Example: page range screening

The following example demonstrates how you can use a partitioning index to enable page range screening on a set of partitions that DB2 needs to examine to satisfy a query predicate.

Suppose that you create table Q1, with partitioning index DATE_IX and DPSI STATE_IX:

```
CREATE TABLESPACE TS1 NUMPARTS 3;

CREATE TABLE Q1 (DATE DATE,
 CUSTNO CHAR(5),
 STATE CHAR(2),
 PURCH_AMT DECIMAL(9,2))
  IN TS1
 PARTITION BY (DATE)
```

```
  (PARTITION 1 ENDING AT ('2002-1-31'),
   PARTITION 2 ENDING AT ('2002-2-28'),
   PARTITION 3 ENDING AT ('2002-3-31'));

CREATE INDEX DATE_IX ON Q1 (DATE) PARTITIONED CLUSTER;

CREATE INDEX STATE_IX ON Q1 (STATE) PARTITIONED;
```

Now suppose that you want to execute the following query against table Q1:

```
SELECT CUSTNO, PURCH_AMT
 FROM Q1
 WHERE STATE = 'CA';
```

Because the predicate is based only on values of a DPSI key (STATE), DB2 must examine all partitions to find the matching rows.

Now suppose that you modify the query in the following way:

```
SELECT CUSTNO, PURCH_AMT
 FROM Q1
 WHERE DATE BETWEEN '2002-01-01' AND '2002-01-31' AND
 STATE = 'CA';
```

Because the predicate is now based on values of a partitioning index key (DATE) and on values of a DPSI key (STATE), DB2 can eliminate the scanning of data partitions 2 and 3, which do not satisfy the query for the partitioning key. This can be determined at bind time because the columns of the predicate are compared to constants.

Now suppose that you use host variables instead of constants in the same query:

```
SELECT CUSTNO, PURCH_AMT
 FROM Q1
 WHERE DATE BETWEEN :hv1 AND :hv2 AND
  STATE = :hv3;
```

DB2 can use the predicate on the partitioning column to eliminate the scanning of unneeded partitions at run time.

Example: page range screening when correlation exists

Writing queries to take advantage of page range screening is especially useful when a correlation exists between columns that are in a partitioning index and columns that are in a DPSI.

For example, suppose that you create table Q2, with partitioning index DATE_IX and DPSI ORDERNO_IX:

```
CREATE TABLESPACE TS2 NUMPARTS 3;

CREATE TABLE Q2 (DATE DATE,
 ORDERNO CHAR(8),
 STATE CHAR(2),
 PURCH_AMT DECIMAL(9,2))
  IN TS2
 PARTITION BY (DATE)
  (PARTITION 1 ENDING AT ('2004-12-31'),
   PARTITION 2 ENDING AT ('2005-12-31'),
   PARTITION 3 ENDING AT ('2006-12-31'));

CREATE INDEX DATE_IX ON Q2 (DATE) PARTITIONED CLUSTER;

CREATE INDEX ORDERNO_IX ON Q2 (ORDERNO) PARTITIONED;
```

Also suppose that the first 4 bytes of each ORDERNO column value represent the four-digit year in which the order is placed. This means that the DATE column and the ORDERNO column are correlated.

To take advantage of page range screening, when you write a query that has the ORDERNO column in the predicate, also include the DATE column in the predicate. The partitioning index on DATE lets DB2 eliminate the scanning of partitions that are not needed to satisfy the query. For example:

```
SELECT ORDERNO, PURCH_AMT
 FROM Q2
 WHERE ORDERNO BETWEEN '2005AAAA' AND '2005ZZZZ' AND
 DATE BETWEEN '2005-01-01' AND '2005-12-31';
```

PSPI

Related concepts:

Page range screening (PAGE_RANGE='Y')

Table space scan access (ACCESSTYPE='R' and PREFETCH='S')

Improving the performance of queries for special situations

You can use special techniques to improve the access paths of queries for certain particular types of data and certain specific types of applications.

Using the CARDINALITY clause to improve the performance of queries with user-defined table function references

The cardinality of a user-defined table function is the number of rows that are returned when the function is invoked. DB2 uses this number to estimate the cost of executing a query that invokes a user-defined table function.

PSPI

The cost of executing a query is one of the factors that DB2 uses when it calculates the access path. Therefore, if you give DB2 an accurate estimate of a user-defined table function's cardinality, DB2 can better calculate the best access path.

You can specify a cardinality value for a user-defined table function by using the CARDINALITY clause of the SQL CREATE FUNCTION or ALTER FUNCTION statement. However, this value applies to all invocations of the function, whereas a user-defined table function might return different numbers of rows, depending on the query in which it is referenced.

To give DB2 a better estimate of the cardinality of a user-defined table function for a particular query, you can use the CARDINALITY or CARDINALITY MULTIPLIER clause in that query. DB2 uses those clauses at bind time when it calculates the access cost of the user-defined table function. Using this clause is recommended only for programs that run on DB2 for z/OS because the clause is not supported on earlier versions of DB2.

Example of using the CARDINALITY clause to specify the cardinality of a user-defined table function invocation

Suppose that when you created user-defined table function TUDF1, you set a cardinality value of 5, but in the following query, you expect TUDF1 to return 30 rows:

```
SELECT *
 FROM TABLE(TUDF1(3)) AS X;
```

Add the CARDINALITY 30 clause to tell DB2 that, for this query, TUDF1 should return 30 rows:

```
SELECT *
 FROM TABLE(TUDF1(3) CARDINALITY 30) AS X;
```

Example of using the CARDINALITY MULTIPLIER clause to specify the cardinality of a user-defined table function invocation

Suppose that when you created user-defined table function TUDF2, you set a cardinality value of 5, but in the following query, you expect TUDF2 to return 30 times that many rows:

```
SELECT *
 FROM TABLE(TUDF2(10)) AS X;
```

Add the CARDINALITY MULTIPLIER 30 clause to tell DB2 that, for this query, TUDF1 should return 5*30, or 150, rows:

```
SELECT *
 FROM TABLE(TUDF2(10) CARDINALITY MULTIPLIER 30) AS X;
```

PSPI

Reducing the number of matching columns

You can discourage the use of a poorer performing index by reducing the index's matching predicate on its leading column.

About this task

PSPI

Consider the example in Figure 20 on page 394, where the index that DB2 picks is less than optimal.

```
CREATE TABLE PART_HISTORY (
    PART_TYPE  CHAR(2),       IDENTIFIES THE PART TYPE
    PART_SUFFIX CHAR(10),     IDENTIFIES THE PART
    W_NOW      INTEGER,       TELLS WHERE THE PART IS
    W_FROM     INTEGER,       TELLS WHERE THE PART CAME FROM
    DEVIATIONS INTEGER,       TELLS IF ANYTHING SPECIAL WITH THIS PART
    COMMENTS        CHAR(254),
    DESCRIPTION     CHAR(254),
    DATE1           DATE,
    DATE2           DATE,
    DATE3           DATE);

  CREATE UNIQUE INDEX IX1 ON PART_HISTORY
    (PART_TYPE,PART_SUFFIX,W_FROM,W_NOW);
  CREATE UNIQUE INDEX IX2 ON PART_HISTORY
    (W_FROM,W_NOW,DATE1);
```

```
+------------------------------------------------------------------------------+
| Table statistics               |       Index statistics   IX1        IX2    |
|--------------------------------+---------------------------------------------|
|CARDF            100,000        |     FIRSTKEYCARDF       1000         50     |
|NPAGES            10,000        |     FULLKEYCARDF     100,000    100,000    |
|                                |     CLUSTERRATIO         99%        99%    |
|                                |     NLEAF               3000       2000    |
|                                |     NLEVELS                3          3    |
|------------------------------------------------------------------------------|
|                column       cardinality    HIGH2KEY     LOW2KEY              |
|                --------     -----------    --------     -------              |
|                Part_type    1000           'ZZ'        'AA'                  |
|                w_now          50            1000         1                   |
|                w_from         50            1000         1                   |
+------------------------------------------------------------------------------+

Q1:
SELECT * FROM PART_HISTORY    --  SELECT ALL PARTS
WHERE PART_TYPE = 'BB'    P1 --  THAT ARE 'BB' TYPES
  AND W_FROM = 3          P2 --  THAT WERE MADE IN CENTER 3
  AND W_NOW = 3           P3 --  AND ARE STILL IN CENTER 3

+------------------------------------------------------------------------------+
|   Filter factor of these predicates.                                         |
|    P1 = 1/1000= .001                                                         |
|    P2 = 1/50  = .02                                                          |
|    P3 = 1/50  = .02                                                          |
|------------------------------------------------------------------------------|
| ESTIMATED VALUES                    |    WHAT REALLY HAPPENS                 |
|                  filter    data     |                        filter   data   |
| index  matchcols  factor    rows    |    index  matchcols   factor   rows    |
|  ix2     2       .02*.02     40     |     ix2     2       .02*.50   1000     |
|  ix1     1       .001       100     |     ix1     1       .001       100     |
+------------------------------------------------------------------------------+
```

Figure 20. Reducing the number of MATCHCOLS

DB2 picks IX2 to access the data, but IX1 would be roughly 10 times quicker. The problem is that 50% of all parts from center number 3 are still in Center 3; they have not moved. Assume that no statistics are available on the correlated columns in catalog table SYSCOLDIST. Therefore, DB2 assumes that the parts from center number 3 are evenly distributed among the 50 centers.

You can get the access path that you want by changing the query. To discourage the use of IX2 for this particular query, you can change the third predicate to be non-indexable.

```
SELECT * FROM PART_HISTORY
  WHERE PART_TYPE = 'BB'
    AND W_FROM = 3
    AND (W_NOW = 3 + 0)      <-- PREDICATE IS MADE NON-INDEXABLE
```

Now index I2 is not picked, because it has only one match column. The preferred index, I1, is picked. The third predicate is a non-indexable predicate, so an index is not used for the compound predicate.

You can make a predicate non-indexable in many ways. The recommended way is to add 0 to a predicate that evaluates to a numeric value or to concatenate an empty string to a predicate that evaluates to a character value.

Indexable	Non-indexable
`T1.C3=T2.C4`	`(T1.C3=T2.C4 CONCAT '')`
`T1.C1=5`	`T1.C1=5+0`

These techniques do not affect the result of the query and cause only a small amount of overhead.

The preferred technique for improving the access path when a table has correlated columns is to generate catalog statistics on the correlated columns. You can do that either by running RUNSTATS or by updating catalog table SYSCOLDIST manually.

PSPI

Rearranging the order of tables in a FROM clause

The order of tables or views in the FROM CLAUSE can affect the access path that DB2 chooses for a SQL query.

About this task

PSPI

If your query performs poorly, it could be because the join sequence is inefficient. You can determine the join sequence within a query block from the PLANNO column in the PLAN_TABLE. If you think that the join sequence is inefficient, try rearranging the order of the tables and views in the FROM clause to match a join sequence that might perform better.

PSPI

Improving outer join processing

You can use a subsystem parameter to improve how DB2 processes an outer join.

About this task

PSPI

OJPERFEH parameter is deprecated beginning in DB2 10, and should always be set to YES.

Procedure

To improve outer join processing:

Set the OJPERFEH subsystem parameter to YES. DB2 takes the following actions, which can improve outer join processing in most cases:

- Does not merge table expressions or views if the parent query block of a table expression or view contains an outer join, and the merge would cause a column in a predicate to become an expression.
- Does not attempt to reduce work file usage for outer joins.
- Uses transitive closure for the ON predicates in outer joins.

Related reference:

OJPERFEH in macro DSN6SPRM (DB2 Installation and Migration)

Deprecated function in DB2 10 (DB2 for z/OS What's New?)

Using a subsystem parameter to optimize queries with IN predicates

You can control how DB2 optimizes IN predicates.

Procedure

PSPI

To control how DB2 optimizes IN predicates:

Set the value of the INLISTP subsystem parameter. When you set the INLISTP parameter to a number *n* that is between 1 and 5000, DB2 optimizes for an IN predicate with up to *n* values. If you set the INLISTP predicate to zero, the optimization is disabled. The default value for the INLISTP subsystem parameter is 50.

When you enable the INLISTP parameter, you enable two primary means of optimizing some queries that contain IN predicates:

- The IN predicate is pushed down from the parent query block into the materialized table expression.
- A correlated IN predicate in a subquery that is generated by transitive closure is moved up to the parent query block.

Related concepts:

How DB2 modifies IN predicates

Predicates generated through transitive closure

Predicate types

Related reference:

INLISTP in macro DSN6SPRM (DB2 Installation and Migration)

Providing more information to DB2 for access path selection

In certain cases you can improve access path selection for SQL statements by providing more information to DB2.

Related tasks:

Modifying catalog statistics to influence access path selection

Managing and preventing access path change

Fetching a limited number of rows

You can specify the fetch clause in a SELECT statement to limit the number of rows in the result table of a query.

About this task

PSPI

In some applications, you execute queries that can return a large number of rows, but you need only a small subset of those rows. Retrieving the entire result table from the query can be inefficient.

Procedure

To limit the number of rows in the result table of a query:

Specify the FETCH FIRST *n* ROWS ONLY clause in the SELECT statement.

Results

DB2 limits the number of rows in the result table of a query to *n* rows.

For distributed queries that use DRDA access, FETCH FIRST *n* ROWS ONLY, DB2 prefetches only *n* rows.

Example

Suppose that you write an application that requires information on only the 20 employees with the highest salaries. To return only the rows of the employee table for those 20 employees, you can write a query as shown in the following example:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC
  FETCH FIRST 20 ROWS ONLY;
```

You can also use FETCH FIRST *n* ROWS ONLY within a subquery.

```
SELECT * FROM EMP
WHERE EMPNO IN (
 SELECT RESPEMP FROM PROJECT
 ORDER BY PROJNO
       FETCH FIRST 3 ROWS ONLY)
```

PSPI

Related concepts:

Interaction between FETCH and OPTIMIZE FOR clauses

Related tasks:

Optimizing retrieval for a small set of rows (DB2 Application programming and SQL)

Related reference:

fetch-first-clause (DB2 SQL)

Minimizing the cost of retrieving few rows

You can improve the performance of applications that retrieve only a small subset of many qualifying rows.

About this task

PSPI

When an application issues a SELECT statement, DB2 assumes that the application retrieves all the qualifying rows. This assumption is most appropriate for batch environments. However, for interactive SQL applications, such as SPUFI, queries commonly define a large potential result set but retrieve only the first few rows.

The OPTIMIZE FOR *n* ROWS clause declares the intent of an application to take one of the following actions:

- Retrieve only a subset of the result set
- Give priority to the retrieval of the first few rows

DB2 uses the OPTIMIZE FOR *n* ROWS clause to choose access paths that minimize the response time for retrieving the first few rows. For distributed queries, the value of *n* determines the number of rows that DB2 sends to the client on each DRDA network transmission.

The OPTIMIZE FOR *n* ROWS clause does the retrieval of all the qualifying rows. However, if you use OPTIMIZE FOR *n* ROWS, the total elapsed time to retrieve all the qualifying rows might be greater than when DB2 optimizes for the entire result set.

OPTIMIZE FOR *n* ROWS is effective only on queries that can be processed incrementally. If the query causes DB2 to gather the entire result set before returning the first row, DB2 ignores the OPTIMIZE FOR *n* ROWS clause. Examples include the following situations:

- The query uses SELECT DISTINCT or a set function distinct, such as COUNT(DISTINCT C1).
- Either GROUP BY or ORDER BY is used, and no index can provide the necessary ordering.
- An aggregate function is used and no GROUP BY clause is used.
- The query uses UNION.

Procedure

To optimize applications the retrieve few of a large qualifying set of rows, use the following approaches:

- To avoid sort operations, specify OPTIMIZE FOR 1 ROW. OPTIMIZE FOR 1 ROW tells DB2 to select an access path that returns the first qualifying row quickly. The result is that DB2 avoids a sort whenever possible. When you specify any value for *n* other than 1, DB2 chooses an access path based on cost, and sort operations remain a possible.

  You can use OPTIMIZE FOR 1 ROW for both local and remote queries. This value does not prevent or restrict block fetch for distributed queries.

  If you continue to encounter sort operations when OPTIMIZE FOR 1 ROW is specified, you can set the value of the OPT1ROWBLOCKSORT subsystem parameter to ENABLE.
- For local queries, specify OPTIMIZE FOR *n* ROWS only in applications that frequently fetch only a small percentage of the total rows in the query result set. For example, an application might read only enough rows to fill a single terminal screen. In such, the application might read the remaining part of the query result set only rarely. For such applications, OPTIMIZE FOR *n* ROWS can result in better performance by causing DB2 to favor SQL access paths that deliver the first *n* rows as fast as possible.
- For remote queries, specify an appropriate value of *n* for the specific situation:

  - Specify a small value for *n* to limit the number of rows that flow across the network on any single transmission.
  - Specify a large value for *n* to improve the performance for receiving a large result set. When you specify a large value, DB2 attempts to send the *n* rows in multiple transmissions. For better performance when retrieving a large result set, do not issue other SQL statements until the entire result set for the query is processed. If retrieval of data for several queries overlaps, DB2 might need to buffer result set data in the DDF address space.
- For a Call Level Interface (CLI) application, you can specify that DB2 uses OPTIMIZE FOR *n* ROWS for all queries. To do that, specify the OPTIMIZEFORNROWS keyword in the initialization file.

Results

The following access path changes are likely when you specify the OPTIMIZE FOR *n* ROWS clause:

- The join method might change. Nested loop join is the most likely choice, because it has a lower cost for the retrieval of only one row.
- An index that matches the ORDER BY clause is more likely to be picked because no sort would be needed for the ORDER BY.
- List prefetch is less likely to be picked.
- Sequential prefetch is less likely to be requested by DB2 because it infers that you want to retrieve only few rows.
- In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if an index created on that outer table gives the ordering needed for the ORDER BY clause.

PSPI

Example

Suppose that you query the employee table regularly to determine the employees with the highest salaries. You might use the following query:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
  FROM EMP
  ORDER BY SALARY DESC;
```

Suppose that an index is defined on column EMPNO, so employee records are ordered by EMPNO. If descending index also exists on the SALARY column, that index is likely to be poorly clustered. To avoid many random synchronous I/O operations, DB2 is likely to use a table space scan, then sort the rows on SALARY. The sort operation is likely to delay the return of the first few rows to the application. However, you might use the following statement to avoid the cost of the sort operation:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
  FROM EMP
  ORDER BY SALARY DESC
  OPTIMIZE FOR 20 ROWS;
```

When you use this statement, DB2 probably uses the SALARY index directly. The query now indicates that you expect to retrieve the salaries of only the 20 most highly paid employees.

Related concepts:

Optimization for large and small result sets (Introduction to DB2 for z/OS)

The effect of the OPTIMIZE FOR n ROWS clause in distributed applications

Interaction between FETCH and OPTIMIZE FOR clauses

Related tasks:

Optimizing retrieval for a small set of rows (DB2 Application programming and SQL)

Improving performance for applications that access distributed data

Related reference:

optimize-clause (DB2 SQL)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Interaction between FETCH and OPTIMIZE FOR clauses

You can specify FETCH and OPTIMIZE FOR clauses in the same query.

In general, if you specify the FETCH clause but not OPTIMIZE FOR clause in a SELECT statement, DB2 optimizes the query as if you had specified the OPTIMIZE FOR clause (that is, OPTIMIZE FOR is implicit when the FETCH clause is used).

When you specify FETCH FIRST *n* ROWS ONLY and OPTIMIZE FOR *m* ROWS, and *m* is less than *n*, DB2 optimizes the query for *m* rows. If *m* is greater than *n*, DB2 optimizes the query for *n* rows.

Suppose that you submit the following SELECT statement:

```
SELECT * FROM EMP
FETCH FIRST 5 ROWS ONLY
OPTIMIZE FOR 20 ROWS;
```

DB2 uses the OPTIMIZE FOR value of 20 rows for access path selection.

Related tasks:

Minimizing the cost of retrieving few rows

Fetching a limited number of rows

Related reference:

fetch-first-clause (DB2 SQL)

optimize-clause (DB2 SQL)

Favoring index access

You can increase the likelihood the DB2 chooses matching index access when other access methods might result in performance problems.

Before you begin

Remember that index access might sometimes be more costly than a table space scan or nonmatching index access. Use these methods only when you are certain that index access is preferable.

About this task

PSPI

DB2 often scans a table space or nonmatching index when statistics indicate that a table is small, even though matching index access is possible. This choice might become a problem in certain cases. For example, a table might be small or empty when statistics are collected. Later, after data is inserted in the table, statistics such as the value of the NPAGES column in the SYSIBM.SYSTABSTATS catalog table might no longer represent the actual volume of data in the table. The result might be that DB2 chooses an inefficient access path because of the inaccurate cost estimate.

Another reason to favor index access might be a database design in which tables contain groups of rows that logically belong together. The rows within each group are accessed in the same sequence every time. The access sequence is based on the primary key. Lock contention can occur when DB2 chooses different access paths for other applications that operate on tables that use this design. Index access can reduce contention and preserve the access sequence that the primary key provides.

Procedure

To favor the use of index access over other access methods, use any of the following approaches:

- Specify the VOLATILE option when you create or alter a table. A table that is defined with the VOLATILE keyword is known as a *volatile table*. When DB2 executes queries that access volatile tables, DB2 chooses index access whenever possible.

  This approach is best for when you want to favor matching index access only for specific tables. It specifically targets queries that access tables that are defined as volatile.

  VOLATILE can favor index usage for single table queries or for the leading table of join queries. However, be aware that VOLATILE might not address the index choice for multi-table joins, or might not adequately influence the table join sequence.
- Specify the value of the NPGTHRSH subsystem parameter. When the value of the NPGTHRSH subsystem parameter is set to a value greater than 0. DB2 compares the value to the NPAGES column in the SYSIBM.SYSTABSTATS catalog table. DB2 uses index access for any query that access a table that has an NPAGES value that is smaller than the NPGTHRSH value. When the value is set to 0, DB2 always uses estimated costs to choose the access paths.

  This option applies to all tables in the subsystem that meet the specified threshold.

Results

DB2 uses real-time statistics values, and favors index access, for access path selection during bind or prepare when a table is defined as VOLATILE or qualifies for the value of the NPGTHRSH subsystem parameter.

Related concepts:

Dynamic collection of index filtering estimates

Related tasks:

Optimizing subsystem parameters for SQL statements by using profiles

Maintaining DB2 database statistics

Related reference:

NPGTHRSH in macro DSN6SPRM (DB2 Installation and Migration)

CREATE TABLE (DB2 SQL)

ALTER TABLE (DB2 SQL)

Statistics used for access path selection

SYSIBM.SYSTABSTATS table (DB2 SQL)

Chapter 29. Improving dynamic SQL performance

You can use several techniques to improve performance for dynamic SQL applications.

About this task

Introductory concepts

Submitting SQL statements to DB2 (Introduction to DB2 for z/OS)

Dynamic SQL applications (Introduction to DB2 for z/OS)

Procedure

To improve the performance of dynamic SQL statement, use any of the following methods:

- Use pureQuery® to execute SQL. With pureQuery you can redirect dynamic queries to become static. You can also use pureQuery to lock in access plans, and choose an execution mode of either static or dynamic.

  For more information about pureQuery, see:Submitting SQL statements to DB2 (Introduction to DB2 for z/OS)
- Enable the caching of dynamic SQL statements. You can use dynamic statement caching to give more static functionality to dynamic SQL statements. Dynamic statement caching saves statements that are already prepared and reuses them when identical statements are called. Dynamic statements can be cached when they have passed the authorization checks if the dynamic statement caching is enabled on your system. You can take any or both of following actions to enable caching for dynamic SQL statements:
  - At the subsystem level, use the CACHEDYN=YES subsystem parameter value to enable the dynamic statement cache. When CACHEDYN=YES is set, applications that issue PREPARE or EXECUTE IMMDEDIATE statements can benefit if the skeleton copy of the statement is found in the global statement cache. If the appropriate conditions are met, the skeleton copy can be copied into the storage for the thread in a process called a *short prepare*. That is, two programs can share the same prepared statement. The application has extra PREPARE operations, but the cost of a full prepare is saved.

    For more information about this approach, see Improving dynamic SQL performance by enabling the dynamic statement cache.
  - GUPI At the package level, use the KEEPDYNAMIC(YES) bind option to enable dynamic SQL statements to be kept after commit points. Any single SQL statement that is bound with the KEEPDYNAMIC(YES) bind option can issue a single PREPARE statement for an SQL statement and omit subsequent prepare operations, even after commit points. To achieve the cost savings of this approach, you must omit the unneeded PREPARE statements from the application program. GUPI

    For more information about this approach, see Methods for keeping prepared statements after commit points.
- Specify appropriate REOPT bind options. You can also use the REOPT bind option to control when DB2 re-optimizes the access path for an SQL statement.

These options can make the SQL statements behave more statically or dynamically. You can use them to customize when and how to optimize your SQL statements.

For more information about REOPT bind options, see Chapter 36, “Reoptimizing SQL statements at run time,” on page 543 and REOPT bind option (DB2 Commands) .

- Specify the DEFER(PREPARE) bind option. DB2 does not prepare a dynamic SQL statement until the statement runs. For dynamic SQL that is used in DRDA access, consider specifying the DEFER(PREPARE) option when you bind or rebind your plans or packages. When a dynamic SQL statement accesses remote data, the PREPARE and EXECUTE statements can be transmitted together over the network together and processed at the remote server. The remote server can then send responses to both statements to the local subsystem together, thereby reducing network traffic.

  For more information about the DEFER(PREPARE) bind option, see REOPT bind option (DB2 Commands) and BIND options for distributed applications
- Eliminate use of the WITH HOLD option for cursors. Defining a cursor WITH HOLD requires sending an extra network message to close the cursor. You can improve performance by eliminating the WITH HOLD option when your application doesn't need to hold cursors open across a commit. This recommendation is particularly true for dynamic SQL applications.

  For more information about the WITH HOLD option for cursors, see:
  - Disable cursor hold behavior for more efficient resource use (DB2 Programming for ODBC)
  - Held and non-held cursors (DB2 Application programming and SQL)
  - DECLARE CURSOR (DB2 SQL)

Related concepts:

Dynamic statement cache enhancements (DB2 for z/OS What's New?)

Related tasks:

Including dynamic SQL in your program (DB2 Application programming and SQL)

Improving performance for applications that access distributed data

Related information:

Dynamic Statement Cache (white paper)

Improving dynamic SQL performance by enabling the dynamic statement cache

The *dynamic statement cache* is a pool in which DB2 saves control structures for prepared SQL statements that can be shared among different threads, plans, and packages. By sharing these control structures, applications can avoid unnecessary preparation processes and thus improve performance.

About this task

Introductory concepts

Submitting SQL statements to DB2 (Introduction to DB2 for z/OS)

Dynamic SQL applications (Introduction to DB2 for z/OS)

Embedded dynamic SQL (Introduction to DB2 for z/OS)

As the DB2 ability to optimize SQL has improved, the cost of preparing a dynamic SQL statement has grown. Applications that use dynamic SQL might be forced to pay this cost more than once. When an application performs a commit operation, it must issue another PREPARE statement if that SQL statement is to be executed again. For a SELECT statement, the ability to declare a cursor WITH HOLD provides some relief but requires that the cursor be open at the commit point. WITH HOLD also causes some locks to be held for any objects that the prepared statement is dependent on. Also, WITH HOLD offers no relief for SQL statements that are not SELECT statements.

DB2 can save prepared dynamic statements in a cache. The cache is a dynamic statement cache pool that all application processes can use to save and retrieve prepared dynamic statements. After an SQL statement has been prepared and is automatically saved in the cache, subsequent prepare requests for that same SQL statement can avoid the costly preparation process by using the statement that is in the cache. Statements that are saved in the cache can be shared among different threads, plans, or packages.

For example, assume that your application program contains a dynamic SQL statement, STMT1, which is prepared and executed multiple times. If you are using the dynamic statement cache when STMT1 is prepared for the first time, it is placed in the cache. When your application program encounters the identical PREPARE statement for STMT1, DB2 uses the already prepared STMT1 that is saved in the dynamic statement cache. The following example shows the identical STMT1 that might appear in your application program:

```
PREPARE STMT1 FROM ...     Statement is prepared and the prepared
EXECUTE STMT1              statement is put in the cache.
COMMIT
⋮
PREPARE STMT1 FROM ...     Identical statement. DB2 uses the prepared
EXECUTE STMT1              statement from the cache.
COMMIT
⋮
```

You must enable the dynamic statement cache before it can be used.

Procedure

To enable the dynamic statement cache to save prepared statements:

Specify YES for the value of the CACHEDYN subsystem parameter.

Related concepts:

Methods for keeping prepared statements after commit points

Dynamic SQL applications (Introduction to DB2 for z/OS)

Related tasks:

Including dynamic SQL in your program (DB2 Application programming and SQL)

Capturing performance information for dynamic SQL statements

Monitoring the dynamic statement cache with READS calls

Calculating the EDM statement cache hit ratio

Enabling dynamic SQL statement caching for ODBC function calls (DB2 Programming for ODBC)

Related reference:

CACHE DYNAMIC SQL field (CACHEDYN subsystem parameter) (DB2 Installation and Migration)

Related information:

Dynamic Statement Cache (white paper)

DB2 statement caching (Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS)

EDM and Dynamic Statement Caching (DB2 for z/OS Best Practices)

Dynamic SQL statements that DB2 can cache

Only certain dynamic SQL statements can be saved in the dynamic statement cache.

The following type of SQL statements can be saved in the dynamic statement cache:
SELECT
UPDATE
INSERT
DELETE
MERGE

Distributed and local SQL statements are eligible to be saved.

Statements that are sent to an accelerator server are eligible to be saved in the cache.

The following types of SQL statement text with SQL bracketed comments can be saved in the dynamic statement cache:

- SQL statement text that begins with SQL bracketed comments that are unnested. No single SQL bracketed comment that begins the statement can be greater than 258 bytes. An example of unnested bracketed comments is /* */ /* */.
- SQL statement text with unnested or nested SQL bracketed comments within the text. An example of nested bracketed comments is /* /* */ */.

Bracketed comments that are in SQL statement source code are saved with the statement text when the SQL statements are placed in the dynamic statement cache, unless other tools remove the bracketed comments before DB2 processes the SQL statement.

SQL statement text that is preceded by SQL simple comments (--) or any other characters besides unnested, bracketed comments is not eligible to be saved in the dynamic statement cache.

Statements in plans or packages that are bound with REOPT(ALWAYS) cannot be saved in the cache. Statements in plans and packages that are bound with REOPT(ONCE) or REOPT(AUTO) can be saved in the cache.

Prepared statements cannot be shared among data sharing members. Because each member has its own EDM pool, a cached statement on one member is not available to an application that runs on another member.

Related tasks:

Including dynamic SQL for varying-list SELECT statements in your program (DB2 Application programming and SQL)

Conditions for statement sharing

If a prepared version of an identical SQL statement already exists in the dynamic statement cache, certain conditions must still be met before DB2 can reuse that prepared statement.

Suppose that S1 and S2 are source statements, and P1 is the prepared version of S1. P1 is in the dynamic statement cache.

The following conditions must be met before DB2 can use statement P1 instead of preparing statement S2:

- S1 and S2 must be identical. The statements must pass a character by character comparison and must be the same length. If the PREPARE statement for either statement contains an ATTRIBUTES clause, DB2 concatenates the values in the ATTRIBUTES clause to the statement string before comparing the strings. That is, if A1 is the set of attributes for S1 and A2 is the set of attributes for S2, DB2 compares S1||A1 to S2||A2. S1 and S2 must be identical if the PREPARE ATTRIBUTES clause CONCENTRATE STATEMENTS WITH LITERALS is not used to request literal constant replacement in S1 and S2.

  If the statement strings are not identical, DB2 cannot use the statement in the cache.

  For example, assume that S1 and S2 are specified as follows:

  ```
  'UPDATE EMP SET SALARY=SALARY+50'
  ```

  In this case, DB2 can use P1 instead of preparing S2.

  However, assume that S1 is specified as follows:

  ```
  'UPDATE EMP SET SALARY=SALARY+50'
  ```

  Assume also that S2 is specified as follows:

  ```
  'UPDATE EMP SET SALARY=SALARY+50 '
  ```

  In this case, DB2 cannot use P1 for S2. DB2 prepares S2 and saves the prepared version of S2 in the cache.
- The authorization ID or role that was used to prepare S1 must be used to prepare S2:
  - When a plan or package has run behavior, the authorization ID is the current SQLID value.

    For secondary authorization IDs:
    - The application process that searches the cache must have the same secondary authorization ID list as the process that inserted the entry into the cache or must have a superset of that list.
    - If the process that originally prepared the statement and inserted it into the cache used one of the privileges held by the primary authorization ID to accomplish the prepare, that ID must either be part of the secondary authorization ID list of the process searching the cache, or it must be the primary authorization ID of that process.
  - When a plan or package has bind behavior, the authorization ID is the plan owner's ID. For a DDF server thread, the authorization ID is the package owner's ID.
  - When a package has define behavior, then the authorization ID is the user-defined function or stored procedure owner.

  – When a package has invoke behavior, then the authorization ID is the authorization ID under which the statement that invoked the user-defined function or stored procedure executed.
  – If the application process has a role associated with it, DB2 uses the role to search the cache instead of the authorization IDs. If the trusted context that associated the role with the application process is defined with the WITH ROLE AS OBJECT OWNER clause, the role value is used as the default for the CURRENT SCHEMA special register and the SQL path.
- When the plan or package that contains S2 is bound, the values of these bind options must be the same as when the plan or package that contains S1 was bound:

  CURRENTDATA
  DYNAMICRULES
  ISOLATION
  SQLRULES
  QUALIFIER
  EXTENDEDINDICATOR
- When S2 is prepared, the values of the following special registers must be the same as when S1 was prepared:

  CURRENT DECFLOAT ROUNDING MODE
  CURRENT DEGREE
  CURRENT RULES
  CURRENT PRECISION
  CURRENT REFRESH AGE
  CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
  CURRENT LOCALE LC_CTYPE
- Two statements must be identical, except for literals. When the PREPARE ATTRIBUTES clause CONCENTRATE STATEMENTS WITH LITERALS is specified and the statements qualify for literal constant replacement, the cached statement (where the literals were already replaced) and the new statement must be identical, except for the literals. To be considered for literal constant replacement, the dynamic SQL statement must not include parameter markers ('?').

  If the first search of the cache does not find an exact match using the original statement text and CONCENTRATE STATEMENTS WITH LITERALS is specified in the ATTRIBUTES clause, the CONCENTRATE STATEMENTS behavior goes into effect. DB2 substitutes the ampersand character ('&') for literal constants in the SQL statement text and continues the cache prepare process, using this new version of the statement text that contains '&'. DB2 searches the cache again to find a matching cached statement that also has '&' substituted for the literal constants. For this second search, the new statement and the cached statement must again pass a character by character statement text comparison, with both statements having '&' for the literals. If that statement text comparison is successful, DB2 determines if the *literal reusability criteria* between the two statements allows for the new statement to share the cached statement.

  For literal reusability, the reusability criteria includes, but is not limited to, the immediate usage context, the literal data type, and the data type size of both the new literal instance and the cached literal instance. If DB2 determines that the new instance of a literal can be reused in place of the cached literal instance, a cached statement that was prepared with the CONCENTRATE STATEMENTS WITH LITERALS clause can be shared by the same SQL statement with a different instance of a literal value. However, that same SQL statement must meet all of the other conditions for sharing the cached statement.

If DB2 determines that the statement with the new literal instance cannot share the cached statement because of incompatible literal reusability criteria, DB2 inserts into the cache a new statement that has both '&' substitution and a different set of literal reusability criteria. This new statement is different from the cached statement, even though both statements have the same statement text with ampersand characters ('&'). Now, both statements are in the cache, but each has different literal reusability criteria that makes these two cached statements unique.

Example 1: Original SQL statement:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 123 (no cache match)
```

After the literals are replaced with '&', the cached statement is as follows:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < &
```

Example 2: Original SQL statement:

```
INSERT INTO TABLE1 (X, Y, Z) VALUES (8,109,29) (no cache match)
```

After the literals are replaced with '&', the cached statement is as follows:

```
INSERT INTO TABLE1 (X, Y, Z) VALUES (&,&,&)
```

Example 3: As an example of the literal reusability criteria, assume that the SELECT statement from example 1 is cached as follows, where column X has data type decimal:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 123 (no cache match)
```

After the literals are replaced with '&', the cached statement is as follows:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < & (+ lit 123 reuse info)
```

Assume that the following new instance of that statement is now being prepared:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 1E2
```

According to the literal reusability criteria that DB2 uses for literal replacement, the literal value 1E2 in the new version of the SELECT statement is not reusable in place of the literal value 123 in the original cached '&' SELECT statement, because the literal value 1E2 does not match the literal data type reusability of the cached statement. Therefore, DB2 does a full cache prepare for this SELECT statement with literal 1E2 and inserts another instance of this '&' SELECT statement into the cache as follows:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < & (+ lit 1E2 reuse info)
```

The new literal reusability information that was used as part of the cache match criteria is also be cached with this instance of the '&' SELECT statement. This literal reusability information is specific to the literal 1E2, making it a new unique entry in the cache even though it is another instance of the same '&' SELECT statement that is cached.

Now, given the two '&' SELECT statements that are cached, let's attempt to prepare the same SELECT statement again but with a different literal value instance from the first two cases as follows:

```
SELECT X, Y, Z FROM TABLE1 WHERE X < 9
```

The DB2 cache behavior, for this scenario with CONCENTRATE STATEMENTS WITH LITERALS in effect, is as follows:

1. DB2 searches the cache, attempting to find an exact match for the new SELECT statement with literal '9' (along with the usual required conditions for a cache match or sharing). No cache match is found.

2. DB2 replaces literal '9' in the SELECT statement with '&' and does a second search of the cache using the new SELECT statement text that has the '&' instead of '9'. DB2 finds two qualifying cached '&' SELECT statements that match. The one for original literal 123 and the second for original literal 1E2.
3. Given the two qualifying '&' SELECT statement cache matches that were found, DB2 continues the cache matching evaluation by using the literal reusability criteria to determine which of the two cache matches is reusable with literal value '9'. In this case, both cached statements are reusable with literal value '9', therefore, simply by order of statement insertion into the cache, cached statement for literal 123 is the first cached statement found that satisfies the literal reusability criteria for the new literal value '9'.
4. DB2 does a short prepare for the SELECT statement with literal '9', using the executable statement structures that are cached for the cached '&' SELECT statement for literal 123.

Exception: If you set the CACHEDYN_FREELOCAL subsystem parameter to 1 and a storage shortage occurs, DB2 frees the cached dynamic statements. In this case, DB2 cannot use P1 instead of preparing statement S2, because P1 no longer exists in the statement cache.

Related concepts:

Reoptimization for statements with replaced literal values

DYNAMICRULES bind option (DB2 Application programming and SQL)

Related reference:

PREPARE (DB2 SQL)

CACHEDYN_FREELOCAL in macro DSN6SPRM (DB2 Installation and Migration)

Capturing performance information for dynamic SQL statements

DB2 maintains statement caching performance statistics records when dynamic statements are cached. The statistics include cache hit ratio and other useful data points that you can use to evaluate the overall performance of your statement caches and statement executions.

Before you begin

- Set the value of the CACHEDYN subsystem parameter to YES.
- Create DSN_STATEMENT_CACHE_TABLE, and the associated LOB and auxiliary tables and indexes. You can find the sample statements for creating these objects in member DSNTESC of the SDSNSAMP library.

About this task

Introductory concepts

Dynamic SQL applications (Introduction to DB2 for z/OS)

When DB2 prepares a dynamic SQL statement, it creates control structures that are used when the statements are executed. When dynamic statement caching is in effect, DB2 stores the control structure associated with each prepared dynamic SQL statement in a storage pool. If that same statement or a matching statement is issued again, DB2 can use the cached control structure, avoiding the expense of preparing the statement again.

Procedure

To externalize the statement cache statistics for performance analysis:

1. To externalize the statement cache statistics for performance analysis:

   ```
   START TRACE(P) CLASS(30) IFCID(316,317,318)
   ```

   IFCID 0316 contains the first 60 bytes of SQL text and statement execution statistics. IFCID 0317 captures the full text of the SQL statement. IFCID 0318 enables the collection of statistics. DB2 begins to collect statistics and accumulates them for the length of time when the trace is on. Stopping the trace resets all statistics.
2. Run the SQL workload that you want to analyze.
3. Issue the following SQL statement in a DSNTEP2 utility job:

   ```
   EXPLAIN STMTCACHE ALL
   ```

   Important: Run the workload and issue the EXPLAIN statement while the traces are still running. If you stop the trace for IFCID 318, all statistics in the dynamic statement cache are reset.
   DB2 extracts all statements from the global cache and writes the statistics information to for all statements in the cache that qualify based on the user's SQLID into the DSN_STATEMENT_CACHE_TABLE. If the SQLID has SYSADM authority, statistics for all statement in the cache are written into the table.
4. Begin your evaluation of the statement cache performance by selecting from the inserted rows from the DSN_STATEMENT_CACHE_TABLE table. For example, you can use the following clauses in your query to identify the *n* queries that have the highest total accumulated CPU time for all the executions of the query during the trace interval:

   ```
   ORDER BY STAT_CPU DESC
   FETCH FIRST n ROWS ONLY;
   ```

   Similarly, you might use the following clauses in your query to identify the top *n* queries that have the highest average CPU time per query execution during the trace interval:

   ```
   SELECT STAT_CPU / STAT_EXEC
   FETCH FIRST n ROWS ONLY;
   ```

What to do next

You can also use optimization tools such as IBM Data Studio or IBM Data Server Manager and InfoSphere® Optim™ Query Workload Tuner to capture and analyze statements from the dynamic statement cache.

Related tasks:

Creating EXPLAIN tables

Monitoring the dynamic statement cache with READS calls

Including dynamic SQL in your program (DB2 Application programming and SQL)

Related reference:

DSN_STATEMENT_CACHE_TABLE

CACHE DYNAMIC SQL field (CACHEDYN subsystem parameter) (DB2 Installation and Migration)

EXPLAIN (DB2 SQL)

DSNTEP2 and DSNTEP4 (DB2 Application programming and SQL)

IBM Data Studio product overview (IBM Data Studio)

DB2 Query Workload Tuner for z/OS

Related information:

IBM Data Server Manager

Invalidation of cached dynamic statements

Various actions and events can invalidate statements in the dynamic statement cache. DB2 uses the full prepare process to generate new access paths for invalid cached dynamic SQL statements.

For example, the following actions can invalidate cached dynamic statements, among others:

- Changing objects referenced by the statement by issuing ALTER, REVOKE, or DROP statements.
- Collecting statistics for objects referenced by the statement with the RUNSTATS utility.

Related tasks:

Invalidating statements in the dynamic statement cache

Related reference:

DSN_STATEMENT_CACHE_TABLE

RUNSTATS (DB2 Utilities)

Invalidating statements in the dynamic statement cache

DB2 invalidates statements in the dynamic statement cache when you run the RUNSTATS utility on objects to which those statements refer. In a data sharing environment, the relevant statements are also invalidated in the cache of other members in the group.

About this task

The invalidation of the cached statements ensures that the next invocations of those statements are fully prepared and use the latest access path changes. The following actions, among others, require that you invalidate cached dynamic statements to get new access paths:

- Statistics collection.
- Online subsystem parameter changes.
- Index changes.
- Changes to profile tables for optimization parameters, production system modeling, or query acceleration thresholds.

This invalidation affects only future PREPARE operations. The next time that one of the invalidated statements is prepared, a new statement is built for the dynamic statement cache. However, any DB2 threads that already retrieved a copy of the statement from the dynamic statement cache before RUNSTATS completes continue to use that copy of the statement.

Important: If you received SQLCODE -904 with reason code 00E70081, this procedure for invalidating statements in the dynamic statement cache does not solve the problem.

Procedure

To invalidate statements in the dynamic statement cache:

Run RUNSTATS on the objects that are referenced by the statements that you want to invalidate. To invalidate cached statements without collecting statistics, specify the UPDATE NONE and REPORT NO options.

Related tasks:

Improving dynamic SQL performance by enabling the dynamic statement cache

Capturing performance information for dynamic SQL statements

Related reference:

RUNSTATS TABLESPACE syntax and options (DB2 Utilities)

Related information:

00E70081 (DB2 Codes)

Methods for keeping prepared statements after commit points

If your program issues the same dynamic SQL statement in different commit scopes, consider specifying that DB2 keeps the prepared versions of these statements after commit points. This behavior can improve performance. By default, DB2 does not keep these statements after commit points.

Introductory concepts

Embedded dynamic SQL (Introduction to DB2 for z/OS)

Dynamic SQL applications (Introduction to DB2 for z/OS)

GUPI

KEEPDYNAMIC(YES) bind option

The KEEPDYNAMIC(YES) bind option lets you hold dynamic statements past a commit point for an application process. An application can issue a PREPARE for a statement once and omit subsequent PREPARE statements for that statement. The following example illustrates an application that is written to use KEEPDYNAMIC(YES).

```
PREPARE STMT1 FROM ...     Statement is prepared.
EXECUTE STMT1
COMMIT
⋮
EXECUTE STMT1              Application does not issue PREPARE.
COMMIT
⋮
EXECUTE STMT1              Again, no PREPARE needed.
COMMIT
```

To understand how the KEEPDYNAMIC bind option works, you need to differentiate between the executable form of a dynamic SQL statement, which is the prepared statement, and the character string form of the statement, which is the statement string.

Relationship between KEEPDYNAMIC(YES) and statement caching: When the dynamic statement cache is not active, and you run an application bound with KEEPDYNAMIC(YES), DB2 saves only the statement string for a prepared statement after a commit point.On a subsequent OPEN, EXECUTE, or DESCRIBE, DB2 must prepare the statement again before performing the requested operation. The following example illustrates this concept.

```
PREPARE STMT1 FROM ...     Statement is prepared and put in memory.
EXECUTE STMT1
COMMIT
⋮
EXECUTE STMT1              Application does not issue PREPARE.
COMMIT                     DB2 prepares the statement again.
⋮
EXECUTE STMT1              Again, no PREPARE needed.
COMMIT
```

When the dynamic statement cache is active, and you run an application bound with KEEPDYNAMIC(YES), DB2 retains a copy of both the prepared statement and the statement string. The prepared statement is cached locally for the application process. In general, the statement is globally cached in the EDM pool, to benefit other application processes. If the application issues an OPEN, EXECUTE, or DESCRIBE after a commit point, the application process uses its local copy of the prepared statement to avoid a PREPARE and a search of the cache. The following example illustrates this process.

```
PREPARE STMT1 FROM ...     Statement is prepared and put in memory.
EXECUTE STMT1
COMMIT
⋮
EXECUTE STMT1              Application does not issue PREPARE.
COMMIT                     DB2 uses the prepared statement in memory.
⋮
EXECUTE STMT1              Again, no PREPARE needed.
COMMIT                     DB2 uses the prepared statement in memory.
⋮
PREPARE STMT1 FROM ...     Again, no PREPARE needed.
COMMIT                     DB2 uses the prepared statement in memory.
```

The local instance of the prepared SQL statement is kept in *ssnm*DBM1 storage until one of the following events occurs:

- The application process ends.
- A rollback operation occurs.
- The application issues an explicit PREPARE statement with the same statement name.

  If the application does issue a PREPARE for the same SQL statement name that has a kept dynamic statement associated with it, the kept statement is discarded and DB2 prepares the new statement.
- The statement is removed from memory because the statement has not been used recently, and the number of kept dynamic SQL statements reaches the subsystem default as set during installation.

Handling implicit prepare errors: If a statement is needed during the lifetime of an application process, and the statement has been removed from the local cache, DB2 might be able to retrieve it from the global cache. If the statement is not in the global cache, DB2 must implicitly prepare the statement again. The application does not need to issue a PREPARE statement. However, if the application issues an OPEN, EXECUTE, or DESCRIBE for the statement, the application must be able to handle the possibility that DB2 is doing the prepare *implicitly*. Any error that occurs during this prepare is returned on the OPEN, EXECUTE, or DESCRIBE.

How KEEPDYNAMIC affects applications that use distributed data: If a requester does not issue a PREPARE after a COMMIT, the package at the DB2 for z/OS server must be bound with KEEPDYNAMIC(YES).If both requester and server are DB2 for z/OS subsystems, the DB2 requester assumes that the KEEPDYNAMIC value for the package at the server is the same as the value for the plan at the requester.

The KEEPDYNAMIC option has performance implications for DRDA clients that specify WITH HOLD on their cursors:

- If KEEPDYNAMIC(NO) is specified, a separate network message is required when the DRDA client issues the SQL CLOSE for the cursor.
- If KEEPDYNAMIC(YES) is specified, the DB2 for z/OS server automatically closes the cursor when SQLCODE +100 is detected, which means that the client does not have to send a separate message to close the held cursor. This reduces network traffic for DRDA applications that use held cursors. It also reduces the duration of locks that are associated with the held cursor.

Note: If one member of a data sharing group has enabled the cache but another has not, and an application is bound with KEEPDYNAMIC(YES), DB2 must implicitly prepare the statement again if the statement is assigned to a member without the cache. This can mean a slight reduction in performance.

Related tasks:

Choosing a RELEASE option

Related reference:

KEEPDYNAMIC bind option (DB2 Commands)

RELEASE bind option (DB2 Commands)

Related information:

EDM and Dynamic Statement Caching (DB2 for z/OS Best Practices)

Chapter 30. Programming for parallel processing

You can significantly reduce the response time for data or processor-intensive queries by taking advantage of the ability of DB2 to initiate multiple parallel operations when it accesses data from a table or index in a partitioned table space.

Related tasks:

Interpreting query parallelism

Tuning parallel processing

Parallel processing

DB2 can initiate multiple parallel operations when it accesses data from a table or index in a partitioned table space.

Query *I/O parallelism* manages concurrent I/O requests for a single query, fetching pages into the buffer pool in parallel. Query I/O parallelism is deprecated and is likely to be removed in a future release. This processing can significantly improve the performance of I/O-bound queries. I/O parallelism is used only when one of the other parallelism modes cannot be used.

Query *CP parallelism* enables true multitasking within a query. A large query can be broken into multiple smaller queries. These smaller queries run simultaneously on multiple processors accessing data in parallel, which reduces the elapsed time for a query.

To expand even farther the processing capacity available for processor-intensive queries, DB2 can split a large query across different DB2 members in a data sharing group. This feature is known as *Sysplex query parallelism*.

DB2 can use parallel operations for processing the following types of operations:

- Static and dynamic queries
- Local and remote data access
- Queries using single table scans and multi-table joins
- Access through an index, by table space scan or by list prefetch
- Sort
- Multi-row fetch, if the cursor is declared as read-only or the statement uses FOR FETCH ONLY

When a view or table expression is materialized, DB2 generates a temporary work file. This type of work file is shareable in CP mode if there is no full outer join case.

Parallelism for partitioned and nonpartitioned table spaces

Parallel operations usually involve at least one table in a partitioned table space. Scans of large partitioned table spaces have the greatest performance improvements where operations can be carried out in parallel.

Both partitioned, nonpartitioned, and partition-by-growth table spaces can take advantage of query parallelism. Parallelism is enabled to include non-clustering indexes. Thus, table access can be run in parallel when the application is bound with DEGREE (ANY) and the table is accessed through a non-clustering index.

Related tasks:

Enabling parallel processing

Disabling query parallelism

Interpreting query parallelism

Related reference:

SET CURRENT DEGREE (DB2 SQL)

CURRENT DEGREE (DB2 SQL)

DEGREE bind option (DB2 Commands)

MAX DEGREE field (PARAMDEG subsystem parameter) (DB2 Installation and Migration)

read-only-clause (DB2 SQL)

Methods of parallel processing

The figures in this topic show how the parallel methods compare with sequential prefetch and with each other.

Assume that a query accesses a table space that has three partitions, P1, P2, and P3. The notations P1, P2, and P3 are partitions of a table space. R1, R2, R3, and so on, are requests for sequential prefetch. The combination P2R1, for example, means the first request from partition 2.

Sequential processing

The following figure shows sequential processing. With *sequential processing*, DB2 takes the three partitions in order, completing partition 1 before starting to process partition 2, and completing 2 before starting 3. Sequential prefetch allows overlap of CP processing with I/O operations, but I/O operations do not overlap with each other. In the example in the following figure, a prefetch request takes longer than the time to process it. The processor is frequently waiting for I/O.

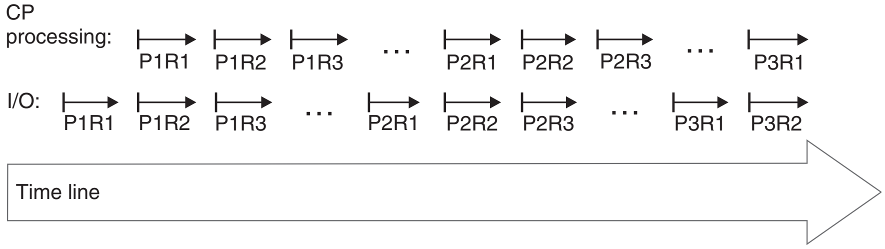


Figure 21. CP and I/O processing techniques. Sequential processing.

Parallel I/O

Query I/O parallelism is deprecated and is likely to be removed in a future release. The following figure shows parallel I/O operations. With *parallel I/O*, DB2 manages data from the three partitions at the same time. The processor processes the first request from each partition, then the second request from each partition, and so on. The processor is not waiting for I/O, but there is still only one processing task.

DB2 can use parallel I/O to improve the efficiency of SELECT and INSERT statements. Parallel INSERT I/O is only available on universal table spaces and partitioned table spaces.

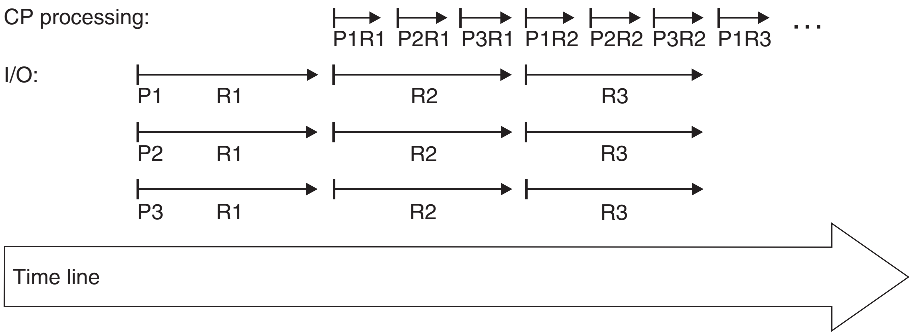


Figure 22. CP and I/O processing techniques. Parallel I/O processing.

Parallel CP processing and sysplex query parallelism

The following figure shows parallel CP processing. With *parallel CP processing*, DB2 can use multiple parallel tasks to process the query. Three tasks working concurrently can greatly reduce the overall elapsed time for data-intensive and processor-intensive queries. The same principle applies for *Sysplex query parallelism*, except that the work can cross the boundaries of a single CPC. Sysplex query parallelism is deprecated and is likely to be removed in a future release.

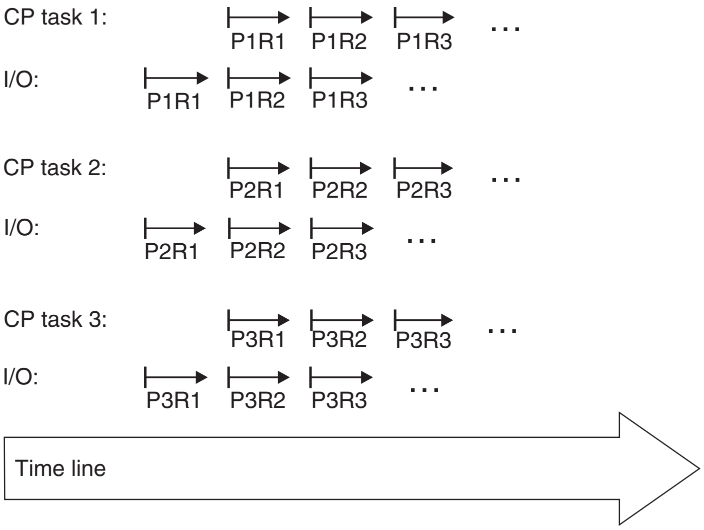


Figure 23. CP and I/O processing techniques. Query processing using CP parallelism. The tasks can be contained within a single CPC or can be spread out among the members of a data sharing group.

Queries that are most likely to take advantage of parallel operations

GUPI

Queries that can take advantage of parallel processing are those queries in which:

- DB2 spends most of the time fetching pages—an I/O-intensive query

  A typical I/O-intensive query is something like the following query, assuming that a table space scan is used on many pages:

```
SELECT COUNT(*) FROM ACCOUNTS
 WHERE BALANCE > 0 AND
 DAYS_OVERDUE > 30;
```

GUPI

- DB2 spends processor time and I/O time to process rows for certain types of queries. Those queries include:

  Queries with intensive data scans and high selectivity
  Those queries involve large volumes of data to be scanned but relatively few rows that meet the search criteria.

  Queries that contain aggregate functions
  Column functions (such as MIN, MAX, SUM, AVG, and COUNT) typically involve large amounts of data to be scanned but return only a single aggregate result.

  Queries that access long data rows
  Those queries access tables with long data rows, and the ratio of rows per page is low (one row per page, for example).

  Queries that require large amounts of central processor time

  GUPI

  Those queries might be read-only queries that are complex, data-intensive, or that involve a sort. For example, A typical processor-intensive query is something like:

```
SELECT MAX(QTY_ON_HAND) AS MAX_ON_HAND,
  AVG(PRICE) AS AVG_PRICE,
  AVG(DISCOUNTED_PRICE) AS DISC_PRICE,
  SUM(TAX) AS SUM_TAX,
  SUM(QTY_SOLD) AS SUM_QTY_SOLD,
  SUM(QTY_ON_HAND - QTY_BROKEN) AS QTY_GOOD,
  AVG(DISCOUNT) AS AVG_DISCOUNT,
  ORDERSTATUS,
  COUNT(*) AS COUNT_ORDERS
FROM   ORDER_TABLE
WHERE SHIPPER = 'OVERNIGHT' AND
      SHIP_DATE < DATE('2006-01-01')
GROUP BY ORDERSTATUS
ORDER BY ORDERSTATUS;
```

  GUPI

Terminology

When the term *task* is used with information about parallel processing, consider the context.For parallel query CP processing or Sysplex query parallelism, a task is an actual z/OS execution unit used to process a query. For parallel I/O processing, a task simply refers to the processing of one of the concurrent I/O streams.

A *parallel group* is the term used to name a particular set of parallel operations. A query can have more than one parallel group, but each parallel group within the query is identified by its own unique ID number.

The *degree of parallelism* is the number of parallel tasks that DB2 determines can be used for the operations on the parallel group. The maximum of parallel operations that DB2 can generate is 254. However, for most queries and DB2 environments, DB2 chooses a lower number.

You might need to limit the maximum number further because more parallel operations consume processor, real storage, and I/O resources. If resource consumption in high in your parallelism environment, use the value of the PARAMDEG subsystem parameter to limit the maximum number of parallel operations.

In a parallel group, an *originating task* is the TCB (SRB for distributed requests) that coordinates the work of all the *parallel tasks*. Parallel tasks are executable units composed of special SRBs, which are called *preemptable* SRBs.

With preemptable SRBs, the z/OS dispatcher can interrupt a task at any time to run other work at the same or higher dispatching priority. For non-distributed parallel work, parallel tasks run under a type of preemptable SRB called a *client* SRB, which lets the parallel task inherit the importance of the originating address space. For distributed requests, the parallel tasks run under a preemptable SRB called an *enclave* SRB.

Related tasks:

Enabling parallel processing

Partitioning for optimal parallel performance

The following are general considerations for how to partition data for the best performance for parallel processing. DB2 does not always select an access path that uses parallelism, regardless of how you partition the data.

About this task

This exercise assumes that the following conditions are true:

- You narrowed the focus to a few critical queries that are running sequentially. It is best to include a mix of I/O-intensive and processor-intensive queries into this initial set. You know how long those queries take now and what your performance objectives for those queries are. Although tuning for one set of queries might not work for all queries, overall performance and throughput can be improved.
- You are optimizing for query-at-a-time operations, and you want a query to use all of the processor and I/O resources available to it.

  When you run many queries at the same time, you might need to increase the number of partitions and the amount of processing power to achieve similar elapsed times.

This information guides you through the following analyses:

1. Determining the nature of the query (what balance of processing and I/O resources it needs)
2. Determining the ideal number of partitions for your table space, based on the nature of the query and on the processor and I/O configuration at your site

Related concepts:

Restrictions for parallelism

Related tasks:

Interpreting query parallelism

Enabling parallel processing

Determining whether queries are I/O- or processor-intensive

How DB2 can a best take advantage of parallel processing for a particular query depends upon whether the query is I/O- or processor-intensive.

Procedure

To determine whether sequential queries are I/O or processor-intensive:

Examine the DB2 accounting reports:

- If the `other read I/O time` is close to the total query elapsed time, then the query is I/O-intensive. `Other read I/O time` is the time that DB2 is waiting for pages to be read in to the buffer pools.
- If `CPU time` is close to the total query elapsed time, then the query is processor-intensive.
- If the processor time is somewhere between 30 and 70 percent of the elapsed time, then the query is pretty well-balanced in terms of CPU and I/O.

Related concepts:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related tasks:

Monitoring parallel operations

Determining the number of partitions for parallel processing

You can calculate the number of partitions that will enable your queries to best take advantage of parallel processing.

About this task

This information provides general guidance for determining the number of partitions. However, you must take into account the I/O subsystem, the nature of the queries that you run, and plan for the data to grow.

If your physical and logical design are not closely tied together, and you can specify any number of partitions, immediately specifying more partitions than you need causes no harm. However, you should start with a reasonable number of partitions because you can always add more partitions later with the ALTER TABLESPACE statement.

You can also create*partition-by-growth* table spaces, which begin as a single-partition table spaces and automatically add partitions as needed to accommodate data growth. Consider creating a partition by growth table space in cases such as a table space with a single table that is expected to become larger than 64 GB, and which does not include a suitable partitioning key.

Consider too the operational complexity of managing many partitions. This complexity might not be as much of an issue at sites that use tools, such as the DB2 Automated Utilities Generator and job scheduler.

In general, the number of partitions falls in a range between the number of CPs and the maximum number of I/O paths to the data. When determining the number of partitions that use a mixed set of processor- and I/O-intensive queries, always choose the largest number of partitions in the range you determine.

Procedure

- For processor-intensive queries, specify, at a minimum, a number that is equal to the number of CPs in the system that you want to use for parallelism, whether you have a single CPC or multiple CPCs in a data sharing group If the query is processor-intensive, it can use all CPs available in the system. If you plan to use Sysplex query parallelism, then choose a number that is close to the total number of CPs (including partial allocation of CPs) that you plan to allocate for decision support processing across the data sharing group. Do not include processing resources that are dedicated to other, higher priority, work.
- For I/O-intensive queries:
  1. Calculate the ratio of elapsed time to processor time.
  2. Multiply that ratio by the number of processors allocated for decision support processing.
  3. Round up the resulting number to determine how many partitions you can use to the best advantage, assuming that these partitions can be on separate devices and have adequate paths to the data.

  This calculation also assumes that you have adequate processing power to handle the increase in partitions. (Which might not be much of an issue with an extremely I/O-intensive query.)

  By partitioning the amount indicated previously, the query is brought into balance by reducing the I/O wait time. If the number of partitions is less than the number of CPs available on your system, increase this number close to the number of CPs available. By doing so, other queries that read this same table, but that are more processor-intensive, can take advantage of the additional processing power.

  Example: Suppose that you have a 10-way CPC and the calculated number of partitions is five. Instead of limiting the table space to five partitions, use 10, to equal the number of CPs in the CPC.

Example configurations for an I/O-intensive query

If the I/O cost of your queries is about twice as much as the processing cost, the optimal number of partitions when run on a 10-way processor is 20 (2 * number of processors). The figure below shows an I/O configuration that minimizes the elapsed time and allows the CPC to run at 100% busy. It assumes the suggested guideline of four devices per control unit and four channels per control unit.[1]

[1] A lower-cost configuration could use as few as two to three channels per control unit shared among all controllers using an ESCON director. However, using four paths minimizes contention and provides the best performance. Paths might also need to be taken offline for service.

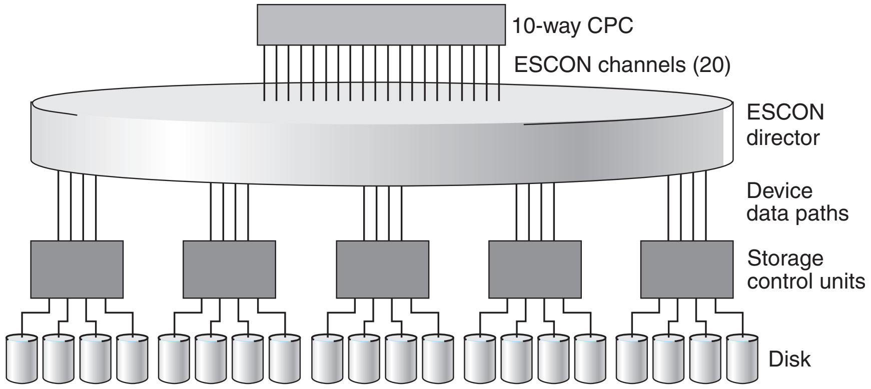


Figure 24. I/O configuration that maximizes performance for an I/O-intensive query

Working with a table space that is already partitioned

You can examine an existing partitioned table space to determine whether parallel processing can be improved.

About this task

Assume that a table space already has 10 partitions and a particular query uses CP parallelism on a 10-way CPC. When you add “other read I/O wait time” (from accounting class 3) and processing time (from accounting class 2), you determine that I/O cost is three times more than the processing cost. In this case, the optimal number of partitions is 30 (three times more I/O paths). However, if you can run on a data sharing group and you add another DB2 subsystem to the group that is running on a 10-way CPC, the I/O configuration that minimizes the elapsed time and allows both CPCs to run at 100% would be 60 partitions.

Making the partitions the same size

The degree of parallelism is influenced by the size of the largest physical partition.

About this task

In most cases, DB2 divides the table space into logical pieces, called *work ranges* to differentiate these from physical pieces, based on the size of the largest physical partition of a given table. Suppose that a table consists of 10 000 pages and 10 physical partitions, the largest of which is 5000 pages. DB2 is most likely to create only two work ranges, and the degree of parallelism would be 2. If the same table has evenly sized partitions of 1000 pages each and the query is I/O-intensive, then ten logical work ranges might be created. This example would result in a degree of parallelism of 10 and reduced elapsed time.

DB2 tries to create equal work ranges by dividing the total cost of running the work by the logical partition cost. This division often has some left over work. In this case, DB2 creates an additional task to handle the extra work, rather than making all the work ranges larger, which would reduce the degree of parallelism.

Procedure

To rebalance partitions that have become skewed:

Reorganize the table space, and specify the REBALANCE keyword on the REORG utility statement.

Working with partitioned indexes

The degree of parallelism for accessing partitioned indexes depends on the nature of the query and on the processor and I/O configuration at your site.

About this task

For an I/O-intensive query, the degree of a parallelism for access to a partitioned index depends on the number of index partitions that are referenced, whereas the degree of parallelism for access to a nonpartitioned index depends on the number of CPs in the system. For a processor-intensive query, the degree of parallelism for both partitioned and nonpartitioned indexes is influenced by the number of CPs in the system.

Enabling parallel processing

Queries cannot take advantage of parallelism unless you enable parallel processing.

Before you begin

DB2 must be running on a central processor complex that contains two or more tightly coupled processors (sometimes called central processors, or CPs). If only one CP is online when the query is bound, DB2 considers only parallel I/O operations. Query I/O parallelism is deprecated and is likely to be removed in a future release.

DB2 considers only parallel I/O operations if you declare a cursor as WITH HOLD and bind the application with RR or RS isolation.

Procedure

To enable parallel processing:

- For static SQL, specify DEGREE(ANY) on BIND or REBIND. This bind option affects static SQL only and does not enable parallelism for dynamic statements.
- For dynamic SQL, set the CURRENT DEGREE special register to 'ANY'.
  - GUPI You can set the special register with the following SQL statement:

    ```
    SET CURRENT DEGREE='ANY';
    ```

    GUPI
  - You can also change the special register default from 1 to ANY for the entire DB2 subsystem by modifying the value of the CDSSRDEF subsystem parameter.

  Setting the special register affects dynamic statements only. It has no effect on your static SQL statements. You must also make sure that parallelism is not disabled for your plan, package, or authorization ID in the RLST.
- If you bind with isolation CS, choose also the option CURRENTDATA(NO), if possible. This option can improve performance in general, but it also ensures that DB2 considers parallelism for ambiguous cursors. If you bind with CURRENTDATA(YES) and DB2 cannot tell if the cursor is read-only, DB2 does not consider parallelism. When a cursor is read-only, it is best to explicitly

specify that the cursor is read-only. You can use the FOR FETCH ONLY or FOR READ ONLY clause on the DECLARE CURSOR statement.

- Specify a virtual buffer pool parallel sequential threshold (VPPSEQT) value that is large enough to provide adequate buffer pool space for parallel processing. If you enable parallel processing, multiple parallel tasks can be activated if DB2 estimates that high elapsed times can be reduced.
- For parallel sorts, allocate sufficient work files to maintain performance. DB2 also considers only parallel I/O operations if you declare a cursor WITH HOLD and bind with isolation RR or RS.
- For complex queries, run the query in parallel within a member of a data sharing group. With Sysplex query parallelism, use the power of the data sharing group to process individual complex queries on many members of the data sharing group.
- Limit the degree of parallelism. To limit the maximum number of parallel tasks that DB2 generates, you can set the value of the PARAMDEG subsystem parameter. If system resources are limited, the best value of MAX DEGREE is 1 - 2 times the number of online CPUs. However, do not change the value PARAMDEG subsystem parameter value to disable parallelism. Instead, use the DEGREE bind parameter or CURRENT DEGREE special register to disable parallelism.

Related tasks:

Tuning parallel processing

Disabling query parallelism

Related reference:

SET CURRENT DEGREE (DB2 SQL)

CURRENT DEGREE (DB2 SQL)

CURRENT DEGREE field (CDSSRDEF subsystem parameter) (DB2 Installation and Migration)

MAX DEGREE field (PARAMDEG subsystem parameter) (DB2 Installation and Migration)

DEGREE bind option (DB2 Commands)

read-only-clause (DB2 SQL)

Restrictions for parallelism

Parallelism is not used for all queries; for some access paths, incurring parallelism overhead makes no sense. Similarly, certain access paths that would reduce the effectiveness of parallelism are removed from consideration when parallelism is enabled.

PSPI

When parallelism is not used

For example, if you are selecting from a temporary table, parallelism is not used. Check the following table to determine whether your query uses any of the access paths that do not allow parallelism.

Table 71. Checklist of parallel modes and query restrictions

If query uses this...	I/O parallelism[1]	CP parallelism	Sysplex parallelism[2]	Comments
Parallel access through RID list (list prefetch and multiple index access)	Yes	Yes	No	Indicated by 'L' in the PREFETCH column of PLAN_TABLE, or an M, MX, MI, or MQ in the ACCESSTYPE column of PLAN_TABLE.
Query blocks that access LOB values.	No	No	No	
Queries that qualify for direct row access	No	No	No	Indicated by 'D' in the PRIMARY_ACCESS_TYPE column of PLAN_TABLE
Materialized views or materialized table expressions at reference time	No	Yes	No	'Yes' for CP applies when there is no full outer join.
Security label column on table	Yes	Yes	No	
Multi-row fetch	Yes	Yes	Yes	Parallelism is available for multi-row fetch if the cursor is read-only or the query contains a FOR FETCH ONLY clause.
Query blocks that access XML values	No	No	No	
Multiple index access to return a DOCID list	No	No	No	Indicated by 'DX', 'DI', or 'DU' in the ACCESSTYPE column of PLAN_TABLE
Outer join result at reference time	No	No	No	
CTE at reference time	No	No	No	
Table function	No	No	No	
Create global temporary table	No	No	No	
Parallel access through IN-list	Yes	Yes	No	Indicated by ACCESSTYPE='N' or 'I' in the PLAN_TABLE.
Parallel access through IN-subquery	No	No	No	Indicated by ACCESSTYPE='N' in the PLAN_TABLE.
A DPSI is used to access the fact table in a star-join	No	No	No	
Correlated subquery block	No	No	No	
Scrollable cursor	No	No	No	
Cursor hold with isolation level 'RR' or 'RS'	Yes	No	No	
Isolation level 'RR' or 'RS'	Yes	Yes	No	
Recursive CTE body	No	No	No	
Hash access	No	No	No	
Range list access	No	No	No	
Reverse index scan	No	No	No	
Table locator	No	No	No	

Table 71. Checklist of parallel modes and query restrictions (continued)

If query uses this...	I/O parallelism [1]	CP parallelism	Sysplex parallelism [2]	Comments
Parallel access through a ROWID column	No	No	No	
Parallel access through a decimal floating point column	No	No	No	
Key range partitioning on timestamp with timezone column	No	No	No	

notes:

1. Query I/O parallelism is deprecated and is likely to be removed in a future release.
2. Sysplex query parallelism is deprecated and is likely to be removed in a future release.

Access paths that are restricted by parallelism

To ensure that you can take advantage of parallelism, DB2 does not select certain access paths when parallelism is enabled. When the plan or package is bound with DEGREE(ANY) or the CURRENT DEGREE special register is set to 'ANY,' DB2

- Does not choose Hybrid joins with SORTN_JOIN=Y.
- Does not transform certain subqueries to joins.

PSPI

Related tasks:

Disabling query parallelism

Related reference:

SET CURRENT DEGREE (DB2 SQL)

CURRENT DEGREE (DB2 SQL)

read-only-clause (DB2 SQL)

Chapter 31. Improving performance for applications that access distributed data

The key to improving the performance of applications that access remote data is to limit the number of network transmissions.

About this task

Introductory concepts

Distributed data (Introduction to DB2 for z/OS)
Distributed data access (Introduction to DB2 for z/OS)

A query that is sent to a remote system can sometimes take longer to execute than the same query, accessing tables of the same size, on the local DB2 subsystem. The principal reasons for this potential increase in execution time are:

- The time required to send messages across the network
- Overhead processing, including startup and communication subsystem session management

Some aspects of overhead processing, for instance, network processing, are not under DB2 control.

Monitoring and tuning performance in a distributed environment is a complex task that requires knowledge of several products.

Procedure

To maximize the performance of an application that accesses distributed data, use the following approaches:

- Write any queries that access distributed data according to the following recommendations to limit the number of messages that these queries send over the network:
  - Reduce the number of columns and rows in the result table by keeping the select lists as short as possible, and use the WHERE, GROUP BY, and HAVING clauses to eliminate unwanted data at the remote server.
  - Specify the FOR FETCH ONLY or FOR READ ONLY clause when possible. Retrieving thousands of rows as a continuous stream is reasonable. Sending a separate message for each one can be significantly slower.

  However, be aware that a query that is sent to a remote subsystem almost always takes longer to execute than the same query that accesses tables of the same size on the local subsystem for the following reasons:
  - Overhead processing, including startup and negotiating session limits (if SNA is used)
  - The time required to send messages across the network
- For any of the following situations, use the OPTIMIZE FOR *n* ROWS clause in your SELECT statements and query result sets from stored procedures:
  - The application fetches only a small number of rows from the query result set.
  - The application fetches a large number of rows from a read-only query.

- The application rarely closes the SQL cursor before fetching the entire query result set.
- The application does not issue statements other than the FETCH statement to the DB2 server while the SQL cursor is open.
- The application does not execute FETCH statements for multiple cursors that are open concurrently and defined with the OPTIMIZE FOR *n* ROWS clause.
- The application does not need to scroll randomly through the data.

The OPTIMIZE FOR *n* ROWS clause limits the number of data rows that the server returns on each DRDA network transmission.

Restriction: This clause has no effect on scrollable cursors.

If you specify 1, 2, or 3 for *n* , DB2 uses the value 16 (instead of *n*) for network blocking and prefetches 16 rows. As a result, network usage is more efficient even though DB2 uses the small value of *n* for query optimization.

For example, the following SQL statement causes DB2 to prefetch 16 rows of the result table even though *n* has a value of 1.

```
SELECT * FROM EMP  OPTIMIZE FOR 1 ROW ONLY;
```

- For queries that have potentially large result tables, but need only a limited number of rows, specify the FETCH FIRST *n* ROWS ONLY clause. This clause limits the number of rows that are returned to a client program.

  For example, suppose that you need only one row of the result table. You can add the FETCH FIRST 1 ROW ONLY clause, as shown in the following example:

  ```
  SELECT * FROM EMP  OPTIMIZE FOR 1 ROW ONLY  FETCH FIRST 1 ROW ONLY;
  ```

  In this case, the FETCH FIRST 1 ROW ONLY clause prevents 15 unnecessary prefetches.
- If your program accesses LOB columns in a remote table, use the following techniques to minimize the number of bytes that are transferred between the client and the server:
  - Use LOB locators instead of LOB host variables.

    If you need to store only a portion of a LOB value at the client, or if your client program manipulates the LOB data but does not need a copy of it, LOB locators are a good choice. When a client program retrieves a LOB column from a server into a locator, DB2 transfers only the 4-byte locator value to the client, not the entire LOB value.
  - Use stored procedure result sets.

    When you return LOB data to a client program from a stored procedure, use result sets rather than passing the LOB data to the client in parameters. Using result sets to return data causes less LOB materialization and less movement of data among address spaces.
  - Set the CURRENT RULES special register to DB2.

    When a DB2 server receives an OPEN request for a cursor, the server uses the value in the CURRENT RULES special register to determine the type of host variables that the associated statement uses to retrieve LOB values. If you specify a value of DB2 for the CURRENT RULES special register before you perform a CONNECT, and the first FETCH statement for the cursor uses a LOB locator to retrieve LOB column values, DB2 lets you use only LOB locators for all subsequent FETCH statements for that column until you close the cursor. If the first FETCH statement uses a host variable, DB2 lets you use only host variables for all subsequent FETCH statements for that column until you close the cursor. However, if you set the value of CURRENT RULES to STD, DB2 lets you use the same open cursor to fetch a LOB column into either a LOB locator or a host variable.

Although a value of STD for the CURRENT RULES special register gives you more programming flexibility when you retrieve LOB data, you get better performance if you use a value of DB2. With the STD option, the server must send and receive network messages for each FETCH statement to indicate whether the data that is being transferred is a LOB locator or a LOB value. With the DB2 option, the server knows the size of the LOB data after the first FETCH, so an extra message about LOB data size is unnecessary. The server can send multiple blocks of data to the requester at one time, which reduces the total time for data transfer.

For example, suppose that a user wants to browse through a large set of employee records and look at pictures of only a few of those employees. At the server, you set the CURRENT RULES special register to DB2. In the application, you declare and open a cursor to select employee records. The application then fetches all picture data into 4-byte LOB locators. Because DB2 knows that 4 bytes of LOB data is returned for each FETCH statement, DB2 can fill the network buffers with locators for many pictures. When a user wants to see a picture for a particular person, the application can retrieve the picture from the server by assigning the value that is referenced by the LOB locator to a LOB host variable. This situation is implemented in the following code:

```
SQL TYPE IS BLOB my_blob[1M];
SQL TYPE IS BLOB AS LOCATOR my_loc;
⋮
FETCH C1 INTO :my_loc;    /* Fetch BLOB into LOB locator  */
⋮
SET :my_blob = :my_loc; /* Assign BLOB to host variable */
```

- Ensure that each cursor meets one of the following conditions when possible, so that DB2 uses block fetch to minimize the number of messages that are sent across the network:
  - The cursor is declared with either the FOR FETCH ONLY or FOR READ ONLY clause.
  - The cursor is a non-scrollable cursor, and the result table of the cursor is read-only.
  - The cursor is a scrollable cursor that is declared as INSENSITIVE, and the result table of the cursor is read-only.
  - The cursor is a scrollable cursor that is declared as SENSITIVE, the result table of the cursor is read-only, and the value of the CURRENTDATA bind option is NO.
  - The result table of the cursor is not read-only, but the cursor is ambiguous, and the value of the CURRENTDATA bind option is NO.

    A cursor is ambiguous when any of the following conditions are true:
    - It is not defined with the clauses FOR FETCH ONLY, FOR READ ONLY, or FOR UPDATE.
    - It is not defined on a read-only result table.
    - It is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement.
    - It is in a plan or package that contains the SQL statements PREPARE or EXECUTE IMMEDIATE.
- For ODBC and JDBC applications, use the rowset parameter to limit the number of rows that are returned from a fetch operation.

If a DRDA requester sends the rowset parameter to a DB2 server, the server performs the following actions:

  - Returns no more than the number of rows in the rowset parameter
  - Returns extra query blocks if the value of the EXTRA BLOCKS SRV field on the DISTRIBUTED DATA FACILITY PANEL 2 installation panel on the server allows extra query blocks to be returned
  - Processes the FETCH FIRST *n* ROWS ONLY clause, if it is specified
  - Does not process the OPTIMIZE FOR *n* ROWS clause

- Use the recommended values for certain bind options. For more information about the recommended bind options, see "BIND options for distributed applications" on page 434.

Related concepts:

How DB2 identifies packages at run time (DB2 Application programming and SQL)

Related tasks:

Designing your application to access distributed data (DB2 Application programming and SQL)

Saving storage when manipulating LOBs by using LOB locators (DB2 Application programming and SQL)

Fetching a limited number of rows

Enabling block fetch for distributed applications

Tuning TCP/IP (DB2 Installation and Migration)

Tuning the VTAM system (DB2 Installation and Migration)

Related reference:

BIND and REBIND options for packages and plans (DB2 Commands)

fetch-first-clause (DB2 SQL)

Remote access and distributed data

DB2 supports remote access between requestor and server relational database management systems (DBMS)

Introductory concepts

Distributed data access (Introduction to DB2 for z/OS)

Remote DB2 access (Introduction to DB2 for z/OS)

Data distribution and Web access (Introduction to DB2 for z/OS)

DB2 uses *DRDA access* for remote connections.

Characteristics of DRDA

With DRDA, the application can remotely bind packages and can execute packages of static or dynamic SQL that have previously been bound at that location. DRDA has the following characteristics and benefits:

- With DRDA access, an application can access data at any server that supports DRDA, not just a DB2 server on a z/OS operating system.
- DRDA supports all SQL features, including user-defined functions, LOBs, stored procedures, and XML data.

- DRDA can avoid multiple binds and minimize the number of binds that are required.
- DRDA supports multiple-row FETCH.

DRDA is the preferred method for remote access with DB2.

Related concepts:

Coding methods for distributed data (DB2 Application programming and SQL)

Serving systems and distributed data

The *serving system* is the DBMS system that runs the remotely bound package.

Introductory concepts

Remote DB2 access (Introduction to DB2 for z/OS)

Distributed data (Introduction to DB2 for z/OS)

Distributed data access (Introduction to DB2 for z/OS)

If you are executing a package on a remote DBMS, then improving performance on the server depends on the nature of the server. If the remote DBMS on which the package executes is another DB2 subsystem, then you can use EXPLAIN information to investigate access path considerations.

Considerations that could affect performance on a remote DB2 server are:

- The maximum number of database access threads that the server allows to be allocated concurrently. (The MAXDBAT subsystem parameter value.) A request can be queued while waiting for an available thread. Making sure that requesters commit frequently can let threads be used by other requesters.
- The Workload Manager priority of database access threads on the remote system. A low priority could impede your application's distributed performance.
- You can manage IDs through DB2 to avoid RACF calls at the server

When DB2 is the server, it is a good idea to activate accounting trace class 7. This provides accounting information at the package level, which can be very useful in determining performance problems.

Related concepts:

Interpreting data access by using EXPLAIN

Managing connection requests from remote applications (Managing Security)

Related tasks:

Setting thread limits for database access threads

Setting performance objectives for distributed workloads by using z/OS Workload Manager

Controlling connections to remote systems (DB2 Administration Guide)

Related reference:

MAX REMOTE ACTIVE field (MAXDBAT subsystem parameter) (DB2 Installation and Migration)

BIND options for distributed applications

Certain bind options can improve the performance of SQL statements that run as part of distributed applications.

Procedure

To improve the performance of applications that access distributed data:

Use the bind options that are shown in the following table:

Table 72. Recommended bind option values for applications that access distributed data

Bind option	Recommended value and actions	Reason
CURRENTDATA	CURRENTDATA(NO)	Use this bind option to force block fetch for ambiguous queries.
ISOLATION	ISOLATION (CS), or any option other than ISOLATION (RR)	When possible, do not bind application plans and packages with ISOLATION(RR). If your application does not need to reference rows that it has already read, another isolation level might reduce lock contention and message overhead during commit processing.
KEEPDYNAMIC	KEEPDYNAMIC(YES)	Use this bind option to improve performance for queries that use cursors that are defined with the WITH HOLD option. With KEEPDYNAMIC(YES), DB2 automatically closes the cursor when no more data exists for retrieval. The client does not need to send a network message to tell DB2 to close the cursor. This option is not needed for clients that close the cursor even if the cursor is defined with the WITH HOLD option.
NODEFER and DEFER	DEFER(PREPARE)	This option reduces network traffic, because the PREPARE and EXECUTE statements and responses are transmitted together.

Table 72. Recommended bind option values for applications that access distributed data (continued)

Bind option	Recommended value and actions	Reason

PKLIST and NOPKLIST	PKLIST
	Specify the package collections for this bind option according to the following recommendations:
	• Reduce the number of packages per collection that DB2 must search. The following example specifies only one package in each collection:
	`PKLIST(S1.COLLA.PGM1, S1.COLLB.PGM2)`
	• Reduce the number of package collections at each location that DB2 must search. The following example specifies only one package collection at each location:
	`PKLIST(S1.COLLA.*, S2.COLLB.*)`
	• Reduce the number of collections that are used for each application. The following example specifies only one collection to search:
	`PKLIST(*.COLLA.*)`
	Requirement: When you specify the DEFER(PREPARE) bind option with DRDA access, the package that contains the statements whose preparation you want to defer must be the first qualifying entry in the package search sequence that DB2 uses.
	For example, assume that the package list for a plan contains two entries:
	`PKLIST(LOCB.COLLA.*, LOCB.COLLB.*)`
	If the intended package is in collection COLLB, ensure that DB2 searches that collection first by executing the following SQL statement:
	`SET CURRENT PACKAGESET = 'COLLB';`
	Alternatively, you can list COLLB first in the PKLIST bind option:
	`PKLIST(LOCB.COLLB.*, LOCB.COLLA.*)`
	For the NODEFER(PREPARE) bind option, the collections in the package list can be in any order, but if the package is not found in the first qualifying PKLIST entry, significant network overhead might result from DB2 searching through the list.
		The order in which you specify package collections in a package list can affect the performance of your application program. When a local instance of DB2 attempts to execute an SQL statement at a remote server, the local DB2 subsystem must determine which package collection the SQL statement is in. DB2 must send a message to the server to request that the server check each collection ID for the SQL statement until the statement is found or no more collection IDs are in the package list. You can reduce the amount of network traffic, and thereby improve performance, by reducing the number of package collections that each server must search.
		As an alternative to specifying the package collections on the PKLIST bind option, you can specify the package collection that is associated with an SQL statement in your application program. Execute the SET CURRENT PACKAGESET statement before you execute an SQL statement to tell DB2 which package collection to search for the statement.

Table 72. Recommended bind option values for applications that access distributed data (continued)

Bind option	Recommended value and actions	Reason
REOPT	Use the following guidelines to decide which option to choose: • Use the REOPT(AUTO) option when the following conditions are true: – You are using the dynamic statement cache. – You want DB2 to decide if a new access path is needed. – Your dynamic SQL statements are executed many times with possibly different input variables. – Similar input variables tend to be executed consecutively. • Use the REOPT(ALWAYS) option on only packages or plans that contain statements that perform poorly because of a bad access path. If you specify REOPT(ALWAYS) when you bind a plan that contains statements that use DB2 private protocol access to access remote data, DB2 prepares those statements twice. • Use the REOPT(ONCE) option when the following conditions are true: – You are using the dynamic statement cache. – You have plans or packages that contain dynamic SQL statements that perform poorly because of access path selection. – Your dynamic SQL statements are executed many times with possibly different input variables. • Use the REOPT(NONE) option when you bind a plan or package that contains statements that use DB2 private protocol access.	Because of performance costs when DB2 reoptimizes the access path at run time, minimize reoptimization when possible.

Related reference:

BIND and REBIND options for packages and plans (DB2 Commands)

Improving performance for SQL statements in distributed applications

In many cases, you can use certain strategies to improve the performance of SQL statements that run on distributed systems.

Procedure

PSPI

To improve SQL statements that access distributed applications, use the following approaches:

- Commit frequently to avoid holding resources at the server.
- Avoid using several SQL statements when one well-tuned SQL statement can retrieve the results that you want. Alternatively, put your SQL statements in a stored procedure, issue your SQL statements at the server through the stored procedure, and return the result. Using a stored procedure creates only one send and receive operation (for the CALL statement) instead of a potential send and receive operation for each SQL statement.

  Depending on how many SQL statements are in your application, using stored procedures can significantly decrease your elapsed time and might decrease your processor costs.
- Use the RELEASE statement and the bind option. The RELEASE statement minimizes the network traffic that is needed to release a remote connection at commit time. For example, if the application has connections to several different servers, specify the RELEASE statement when the application has completed processing for each server. The RELEASE statement does not close cursors, release any resources, or prevent further use of the connection until the COMMIT is issued. It just makes the processing at COMMIT time more efficient.

  The bind option DISCONNECT(EXPLICIT) destroys all remote connections for which RELEASE was specified.
- Consider using the COMMIT ON RETURN YES clause of the CREATE PROCEDURE statement to indicate that DB2 should issue an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. Using the clause can reduce the length of time locks are held and can reduce network traffic. With COMMIT ON RETURN YES, any updates made by the client before calling the stored procedure are committed with the stored procedure changes.
- Set the value of the CURRENT RULES special register to DB2. When requesting LOB data, set the CURRENT RULES special register to DB2 instead of to STD before performing a CONNECT. A value of DB2, which is the default, can offer performance advantages. When a DB2 for z/OS server receives an OPEN request for a cursor, the server uses the value in the CURRENT RULES special register to determine whether the application intends to switch between LOB values and LOB locator values when fetching different rows in the cursor. If you specify a value of DB2 for CURRENT RULES, the application indicates that the first FETCH request specifies the format for each LOB column in the answer set and that the format does not change in a subsequent FETCH request. However, if you set the value of CURRENT RULES to STD, the application intends to fetch a LOB column into either a LOB locator host variable or a LOB host variable.

  Although a value of 'STD for CURRENT RULES gives you more programming flexibility when you retrieve LOB data, you can get better performance if you use a value of DB2. With the STD option, the server does not block the cursor. With the DB2 option, it might block the cursor where it is possible to do so.

GUPI

Related concepts:

COMMIT and ROLLBACK statements in a stored procedure (DB2 Application programming and SQL)

LOB and XML data and its effect on block fetch for DRDA

Related reference:

RELEASE (connection) (DB2 SQL)

CREATE PROCEDURE (DB2 SQL)

CONNECT (DB2 SQL)

CURRENT RULES (DB2 SQL)

SET CURRENT RULES (DB2 SQL)

The effect of the OPTIMIZE FOR *n* ROWS clause in distributed applications

You can specify the OPTIMIZE FOR *n* ROWS clause to improve the performance of certain queries. For queries that access distributed data, this clause can have a significant performance impact because it helps limit the amount of data that is sent over the network. It also limits the number of network transmissions.

When you specify the OPTIMIZE FOR *n* ROWS clause in your query, the number of rows that DB2 transmits on each network transmission depends on the following factors:

- If *n* rows of the SQL result set fit within a single DRDA query block, a DB2 server can send *n* rows to any DRDA client. In this case, DB2 sends *n* rows in each network transmission until the entire query result set is returned.
- If *n* rows of the SQL result set exceed a single DRDA query block, the number of rows that are contained in each network transmission depends on the client's DRDA software level and configuration. The following conditions apply:
  - If the client does not support extra query blocks, the DB2 server automatically reduces the value of *n* to match the number of rows that fit within a DRDA query block.
  - If the client supports extra query blocks, the DRDA client can choose to accept multiple DRDA query blocks in a single data transmission. DRDA allows the client to establish an upper limit on the number of DRDA query blocks in each network transmission.

    The number of rows that a DB2 server sends is the smaller of the following values:

    - *n* rows
    - the number of rows that fit within the maximum number of extra DRDA query blocks that the DB2 server returns to a client in a single network transmission. (This value is specified in the EXTRA BLOCKS SRV field on installation panel DSNTIP5 at the DB2 server.)
    - the number of rows that fit within the client's extra query block limit, which is obtained from the DDM MAXBLKEXT parameter that is received from the client. (When DB2 acts as a DRDA client, the DDM MAXBLKEXT parameter is set to the value of EXTRA BLOCKS REQ on installation panel DSNTIP5.)

Depending on the value that you specify for *n*, the OPTIMIZE FOR *n* ROWS clause can improve performance in the following ways:

- If *n* is less than the number of rows that fit in the DRDA query block, OPTIMIZE FOR *n* ROWS can improve performance by preventing the DB2 server from fetching rows that might never be used by the DRDA client application.
- If *n* is greater than the number of rows that fit in a DRDA query block, OPTIMIZE FOR *n* ROWS lets the DRDA client request multiple blocks of query data on each network transmission. This use of OPTIMIZE FOR *n* ROWS can significantly improve elapsed time for applications that download large amounts of data.

Although the OPTIMIZE FOR *n* ROWS clause can improve performance, this same function can degrade performance if you do not use it properly. The following examples demonstrate the performance problems that can occur when you do not use this clause judiciously.

In the following figure, the DRDA client opens a cursor and fetches rows from the cursor. At some point before all rows in the query result set are returned, the application issues an SQL INSERT statement.

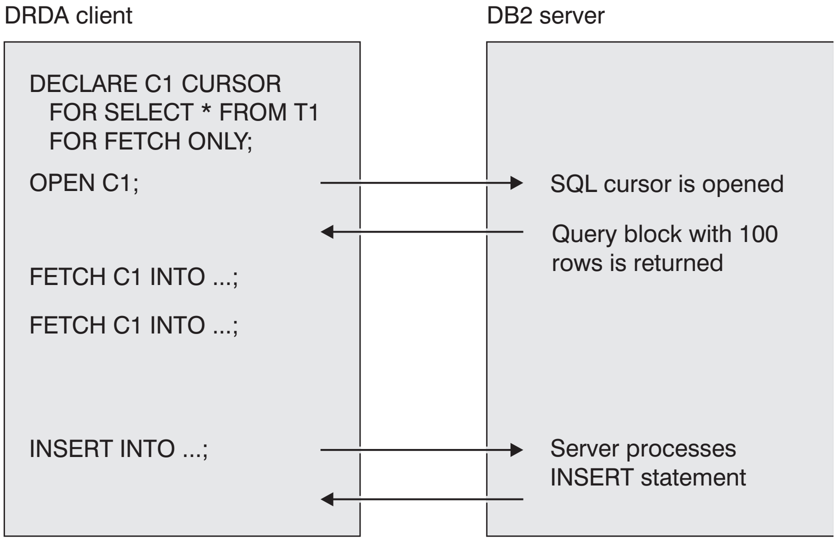


Figure 25. Message flows without the OPTIMIZE FOR n ROWS clause

In this case, DB2 uses normal DRDA message blocking, which has the following advantages over the message blocking that is used for the OPTIMIZE FOR *n* ROWS clause:

- If the application issues an SQL statement other than FETCH (for example, an INSERT statement in this case), the DRDA client can transmit the SQL statement immediately, because the DRDA connection is not in use after the SQL OPEN.
- The DRDA query block size places an upper limit on the number of rows that are fetched unnecessarily. If the SQL application closes the cursor before fetching all the rows in the query result set, the server fetches only the number of rows that fit in one query block, which is 100 rows of the result set.

In the following figure, the DRDA client opens a cursor and fetches rows from the cursor by using OPTIMIZE FOR *n* ROWS clause. Both the DRDA client and the DB2 server are configured to support multiple DRDA query blocks. At some time before the end of the query result set, the application issues an SQL INSERT.

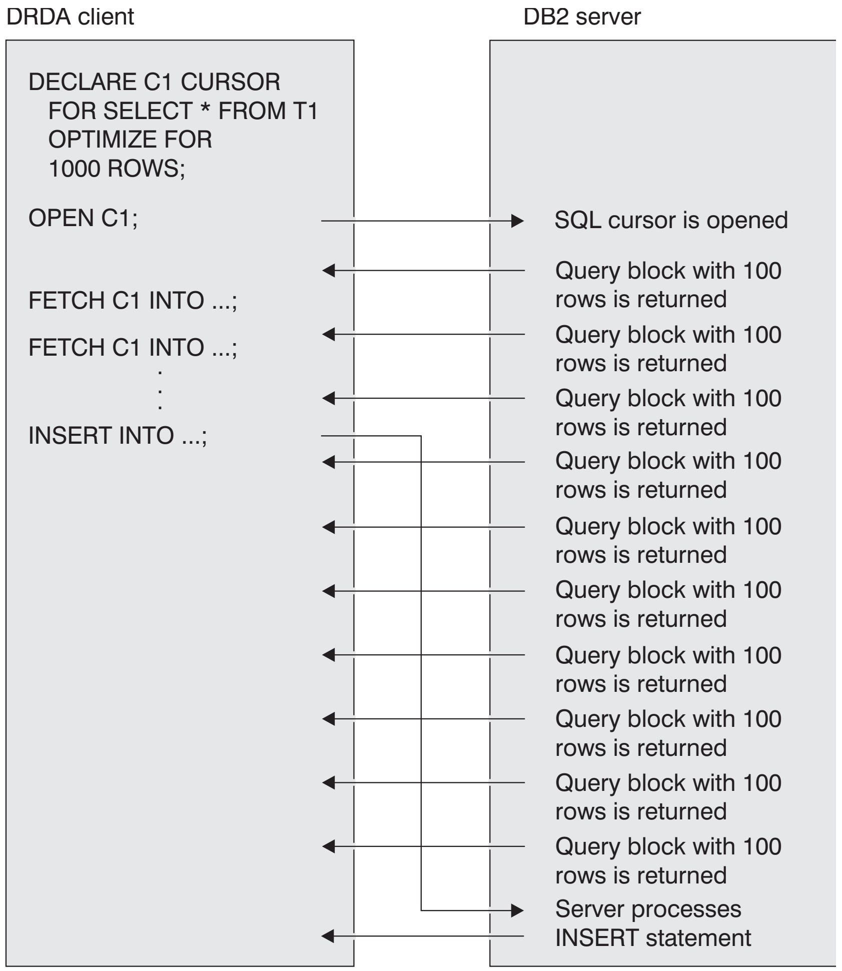

DRDA client
DB2 server
DECLARE C1 CURSOR
FOR SELECT * FROM T1
OPTIMIZE FOR
1000 ROWS;
OPEN C1;
FETCH C1 INTO ...;
FETCH C1 INTO ...;
INSERT INTO ...;
SQL cursor is opened
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Query block with 100 rows is returned
Server processes INSERT statement

Figure 26. Message flows with the OPTIMIZE FOR 1000 ROWS clause

Because the query uses the OPTIMIZE FOR *n* ROWS clause, the DRDA connection is not available when the SQL INSERT is issued. The connection is still being used to receive the DRDA query blocks for 1000 rows of data. This situation causes the following performance problems:

- Application elapsed time can increase if the DRDA client waits for a large query result set to be transmitted before the DRDA connection can be used for other SQL statements. In this example, the SQL INSERT statement is delayed because of a large query result set.
- If the application closes the cursor before fetching all the rows in the SQL result set, the server might fetch a large number of rows unnecessarily.

Related concepts:

Optimization for large and small result sets (Introduction to DB2 for z/OS)

Optimizing for very large result sets for DRDA

Related tasks:

Minimizing the cost of retrieving few rows

Optimizing retrieval for a small set of rows (DB2 Application programming and SQL)

Optimizing for small results sets for DRDA

Related reference:

optimize-clause (DB2 SQL)

TCP/IP KEEPALIVE field (TCPKPALV subsystem parameter) (DB2 Installation and Migration)

EXTRA BLOCKS SRV field (EXTRASRV subsystem parameter) (DB2 Installation and Migration)

Fast implicit close

When you specify the FETCH FIRST *n* ROWS ONLY clause in a distributed query, DB2 might use a fast implicit close to improve performance. *Fast implicit close* is the process of DB2 closing a cursor after prefetching the *n* row or when no more rows are to be returned.

Fast implicit close can improve query performance, because it saves an additional network transmission between the client and the server.

DB2 uses fast implicit close when all of the following conditions are true:

- The query uses limited block fetch.
- The query does not retrieve any LOBs.
- The cursor is not a scrollable cursor.
- Either of the following conditions is true:
  - The cursor is defined with the WITH HOLD option, and the package or plan that contains the cursor is bound with the KEEPDYNAMIC(YES) option.
  - The cursor is not defined with the WITH HOLD option.

Related concepts:

Block fetch (Introduction to DB2 for z/OS)

Limited block fetch

Related tasks:

Enabling block fetch for distributed applications

Related reference:

fetch-first-clause (DB2 SQL)

Enabling block fetch for distributed applications

Block fetch can significantly decrease the number of messages sent across the network.

About this task

Introductory concepts

Block fetch (Introduction to DB2 for z/OS)

With *block fetch*, DB2 groups the rows that are retrieved by an SQL query into as large a “block” of rows as can fit in a message buffer. DB2 then transmits the block over the network, without requiring a separate message for each row.

DB2 can use different types of block fetch:

- Limited block fetch
- Continuous block fetch

To enable limited or continuous block fetch, DB2 must determine that the cursor is not used for updating or deleting. Block fetch is used only with cursors that do not update or delete data.

DB2 triggers block fetch for static SQL only when it can detect that no updates or deletes are in the application. For dynamic statements, because DB2 cannot detect what follows in the program, the decision to use block fetch is based on the declaration of the cursor.

DB2 does not use continuous block fetch if the following conditions are true:

- The cursor is referred to in the statement DELETE WHERE CURRENT OF elsewhere in the program.
- The cursor statement appears that it can be updated at the requesting system. (DB2 does not check whether the cursor references a view at the server that cannot be updated.)

Procedure

To ensure that DB2 uses block fetch:

GUPI

The easiest way to indicate that the cursor does not modify data is to add the FOR FETCH ONLY or FOR READ ONLY clause to the query in the DECLARE CURSOR statement as in the following example:

```
EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8A10.EMP
    WHERE WORKDEPT = 'D11'
    FOR FETCH ONLY
END-EXEC.
```

If you do not use FOR FETCH ONLY or FOR READ ONLY, DB2 still uses block fetch for the query if the following conditions are true:

- The cursor is a non-scrollable cursor, and the result table of the cursor is read-only. This applies to static and dynamic cursors except for read-only views.
- The cursor is a scrollable cursor that is declared as INSENSITIVE, and the result table of the cursor is read-only.
- The cursor is a scrollable cursor that is declared as SENSITIVE, the result table of the cursor is read-only, and the value of bind option CURRENTDATA is NO.
- The result table of the cursor is not read-only, but the cursor is ambiguous, and the value of bind option CURRENTDATA is NO. A cursor is ambiguous when:
  - It is not defined with the clauses FOR FETCH ONLY, FOR READ ONLY, or FOR UPDATE OF.
  - It is not defined on a read-only result table.
  - It is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement.
  - It is in a plan or package that contains the SQL statements PREPARE or EXECUTE IMMEDIATE.

Results

The following tables summarize the conditions under which a DB2 server uses block fetch.

The following table shows the conditions for a non-scrollable cursor.

Table 73. Effect of CURRENTDATA and cursor type on block fetch for a non-scrollable cursor

Isolation level	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	Yes	Read-only	Yes
		Updatable	No
		Ambiguous	No
	No	Read-only	Yes
		Updatable	No
		Ambiguous	Yes
UR	Yes	Read-only	Yes
	No	Read-only	Yes

The following table shows the conditions for a scrollable cursor that is not used to retrieve a stored procedure result set.

Table 74. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is not used for a stored procedure result set

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	No
			Updatable	No
			Ambiguous	No
		No	Read-only	Yes
			Updatable	No
			Ambiguous	Yes
UR	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes

The following table shows the conditions for a scrollable cursor that is used to retrieve a stored procedure result set.

Table 75. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is used for a stored procedure result set

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	No
		No	Read-only	Yes
UR	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes

GUPI

Related concepts:

Problems with ambiguous cursors

Related tasks:

Choosing an ISOLATION option

Related reference:

read-only-clause (DB2 SQL)

DECLARE CURSOR (DB2 SQL)

Continuous block fetch

In terms of response time, continuous block fetch is most efficient for larger result sets because fewer messages are transmitted from the requester to retrieve the entire result set and because overlapped processing is performed at the requester and the server.

Introductory concepts

Block fetch (Introduction to DB2 for z/OS)

However, continuous block fetch uses more networking resources than limited block fetch. When networking resources are critical, use limited block fetch to run applications.

The requester can use both forms of blocking at the same time and with different servers.

If an application is doing read-only processing and can use continuous block fetch, the sequence goes like this:

1. The requester sends a message to open a cursor and begins fetching the block of rows at the server.
2. The server sends back a block of rows and the requester begins processing the first row.
3. The server continues to send blocks of rows to the requester, without further prompting. The requester processes the second and later rows as usual, but fetches them from a buffer on the requester's system.

For DRDA, only one conversation is used, and it must be made available to the other SQL statements that are in the application. Thus, the server usually sends back a subset of all the rows. The number of rows that the server sends depends on the following factors:

- The size of each row
- The number of extra blocks that are requested by the requesting system compared to the number of extra blocks that the server returns

  For a DB2 for z/OS requester, the EXTRA BLOCKS REQ field on installation panel DSNTIP5 determines the maximum number of extra blocks requested. For a DB2 for z/OS server, the EXTRA BLOCKS SRV field on installation panel DSNTIP5 determines the maximum number of extra blocks allowed.

  Example: Suppose that the requester asks for 100 extra query blocks and that the server allows only 50. The server returns no more than 50 extra query blocks. The server might choose to return fewer than 50 extra query blocks for any number of reasons that DRDA allows.
- Whether continuous block fetch is enabled, and the number of extra rows that the server can return if it regulates that number.

  To enable continuous block fetch for DRDA and to regulate the number of extra rows sent by a DB2 for z/OS server, you must use the OPTIMIZE FOR n ROWS clause on your SELECT statement.

If you want to use continuous block fetch for DRDA, have the application fetch all the rows of the cursor before doing any other SQL. Fetching all the rows first prevents the requester from having to buffer the data, which can consume a lot of storage. Choose carefully which applications should use continuous block fetch for DRDA.

Related concepts:

Limited block fetch

Optimizing for very large result sets for DRDA

Related reference:

optimize-clause (DB2 SQL)

Limited block fetch

Limited block fetch guarantees the transfer of a minimum amount of data in response to each request from the requesting system.

With limited block fetch, a single conversation is used to transfer messages and data between the requester and server for multiple cursors. Processing at the requester and server is synchronous. The requester sends a request to the server, which causes the server to send a response back to the requester. The server must then wait for another request to tell it what should be done next.

Related concepts:

Block fetch (Introduction to DB2 for z/OS)

Continuous block fetch

Block fetch with scrollable cursors for DRDA

When a DB2 for z/OS requester uses a scrollable cursor to retrieve data from a DB2 for z/OS server, the following conditions are true.

- The requester never requests more than 64 rows in a query block, even if more rows fit in the query block. In addition, the requester never requests extra query blocks. This is true even if the setting of field EXTRA BLOCKS REQ in the

DISTRIBUTED DATA FACILITY PANEL 2 installation panel on the requester allows extra query blocks to be requested.

- The requester discards rows of the result table if the application does not use those rows.

  Example: If the application fetches row *n* and then fetches row *n*+2, the requester discards row *n*+1.

  The application gets better performance for a blocked scrollable cursor if it mostly scrolls forward, fetches most of the rows in a query block, and avoids frequent switching between FETCH ABSOLUTE statements with negative and positive values.
- If the scrollable cursor does not use block fetch, the server returns one row for each FETCH statement.

LOB and XML data and its effect on block fetch for DRDA

For a non-scrollable blocked cursor, the server sends all the non-LOB and non-XML data columns for a block of rows in one message, including LOB locator values.

As each row is fetched by the application, the requester obtains the non-LOB data columns directly from the query block. If the row contains non-null and non-zero length LOB values, those values are retrieved from the server at that time. This behavior limits the impact to the network by pacing the amount of data that is returned at any one time. If all LOB data columns are retrieved into LOB locator host variables or if the row does not contain any non-null or non-zero length LOB columns, then the whole row can be retrieved directly from the query block.

For a scrollable blocked cursor, the LOB data columns are returned at the same time as the non-LOB and non XML data columns. When the application fetches a row that is in the block, a separate message is not required to get the LOB columns.

Optimizing for very large result sets for DRDA

Enabling a DB2 client to request multiple query blocks on each transmission can reduce network activity and improve performance significantly for applications that use DRDA access to download large amounts of data.

You can specify a large value of *n* in the OPTIMIZE FOR *n* ROWS clause of a SELECT statement to increase the number of DRDA query blocks that a DB2 server returns in each network transmission for a non-scrollable cursor. If *n* is greater than the number of rows that fit in a DRDA query block, OPTIMIZE FOR *n* ROWS lets the DRDA client request multiple blocks of query data on each network transmission instead of requesting a new block when the first block is full. This use of OPTIMIZE FOR *n* ROWS is intended only for applications in which the application opens a cursor and downloads great amounts of data. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

Recommendation: Because the application SQL uses only one conversation, do not try to do other SQL work until the entire answer set is processed. If the application issues another SQL statement before the previous statement's answer set has been received, DDF must buffer them in its address space. You can buffer up to 10 MB in this way.

Because specifying a large number of network blocks can saturate the network, limit the number of blocks according to what your network can handle. You can limit the number of blocks used for these large download operations. When the client supports extra query blocks, DB2 chooses the **smallest** of the following values when determining the number of query blocks to send:

- The number of blocks into which the number of rows (*n*) on the OPTIMIZE clause can fit. For example, assume you specify 10000 rows for *n*, and the size of each row that is returned is approximately 100 bytes. If the block size used is 32 KB (32768 bytes), the calculation is as follows:

  ```
  (10000 * 100) / 32768 = 31 blocks
  ```
- The DB2 server value for the EXTRA BLOCKS SRV field on installation panel DSNTIP5. The maximum value that you can specify is 100.
- The client's extra query block limit, which is obtained from the DRDA MAXBLKEXT parameter received from the client. When DB2 for z/OS acts as a DRDA client, you set this parameter at installation time with the EXTRA BLOCKS REQ field on installation panel DSNTIP5. The maximum value that you can specify is 100. DB2 Connect sets the MAXBLKEXT parameter to -1 (unlimited).

If the client does not support extra query blocks, the DB2 server on z/OS automatically reduces the value of *n* to match the number of rows that fit within a DRDA query block.

Recommendation for cursors that are defined WITH HOLD: Do not set a large number of query blocks for cursors that are defined WITH HOLD. If the application commits while there are still a lot of blocks in the network, DB2 buffers the blocks in the requester's memory (the *ssnm*DIST address space if the requester is a DB2 for z/OS) before the commit can be sent to the server.

Related concepts:

The effect of the OPTIMIZE FOR n ROWS clause in distributed applications

Related reference:

optimize-clause (DB2 SQL)

EXTRA BLOCKS SRV field (EXTRASRV subsystem parameter) (DB2 Installation and Migration)

Optimizing for small results sets for DRDA

When a client does not need all the rows from a potentially large result set, preventing the DB2 server from returning all the rows for a query can reduce network activity and improve performance significantly for DRDA applications.

About this task

GUPI

You can use either the OPTIMIZE FOR *n* ROWS clause or the FETCH FIRST *n* ROWS ONLY clause of a SELECT statement to limit the number of rows returned to a client program.

Using OPTIMIZE FOR *n* ROWS: When you specify OPTIMIZE FOR *n* ROWS and *n* is less than the number of rows that fit in the DRDA query block (default size on z/OS is 32 KB), the DB2 server prefetches and returns only as many rows as fit into the query block. For example, if the client application is interested in seeing only one screen of data, specify OPTIMIZE FOR *n* ROWS, choosing a small number for *n*, such as 3 or 4. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

Using FETCH FIRST *n* ROWS ONLY: The FETCH FIRST *n* ROWS ONLY clause does not affect network blocking. If FETCH FIRST *n* ROWS ONLY is specified and OPTIMIZE FOR *n* ROWS is not specified, DB2 uses the FETCH FIRST value to optimize the access path. However, DRDA does not consider this value when it determines network blocking.

When both the FETCH FIRST *n* ROWS ONLY clause and the OPTIMIZE FOR *n* ROWS clause are specified, the value for the OPTIMIZE FOR *n* ROWS clause is used for access path selection.

Example: Suppose that you submit the following SELECT statement:

```
SELECT * FROM EMP
FETCH FIRST 5 ROWS ONLY
OPTIMIZE FOR 20 ROWS;
```

The OPTIMIZE FOR value of 20 rows is used for network blocking and access path selection.

When you use FETCH FIRST *n* ROWS ONLY, DB2 might use a *fast implicit close.* Fast implicit close means that during a distributed query, the DB2 server automatically closes the cursor when it prefetches the *n*th row if FETCH FIRST *n* ROWS ONLY is specified or when there are no more rows to return. Fast implicit close can improve performance because it can save an additional network transmission between the client and the server.

DB2 uses fast implicit close when the following conditions are true:

- The query uses limited block fetch.
- The query retrieves no LOBs.
- The query retrieves no XML data.
- The cursor is not a scrollable cursor.
- Either of the following conditions is true:
  - The cursor is declared WITH HOLD, and the package or plan that contains the cursor is bound with the KEEPDYNAMIC(YES) option.
  - The cursor is declared WITH HOLD and the DRDA client passes the QRYCLSIMP parameter set to SERVER MUST CLOSE, SERVER DECIDES, or SERVER MUST NOT CLOSE.
  - The cursor is not defined WITH HOLD.

When you use FETCH FIRST *n* ROWS ONLY and DB2 does a fast implicit close, the DB2 server closes the cursor after it prefetches *n* rows, or when there are no more rows.

GUPI

Related concepts:

Optimization for large and small result sets (Introduction to DB2 for z/OS)

Related tasks:

Optimizing retrieval for a small set of rows (DB2 Application programming and SQL)

Minimizing the cost of retrieving few rows

Related reference:

- optimize-clause (DB2 SQL)
- fetch-first-clause (DB2 SQL)

Data encryption security options

Data encryption security options provide added security for the security-sensitive data that an application requests from the system. However, the encryption options can also have a negative impact on performance.

About this task

The following encryption options have a larger performance cost than other options:

- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data

Recommendation: To maximize performance of requester systems, use the minimum level of security that is required by the sensitivity of the data.

Chapter 32. Best practices for XML performance in DB2

By observing certain best practices you can help to improve the performance of XML data that is stored in DB2 for z/OS.

Choose the granularity of XML documents carefully

When you design your XML application, and your XML document structure in particular, you might have a choice to define which business data is kept together in a single XML document.

GUPI

For example, each XML document in the sample department data contains information for one department.

```
CREATE TABLE DEPT(UNITID CHAR(8), DEPTDOC XML);
```

unitID	deptdoc

WWPR	<dept deptID="PR27">
	   <employee id="901">
	      <name>Jim Qu</name>
	      <phone>408 555 1212</phone>
	   </employee>
	   <employee id="902">
	      <name>Peter Pan</name>
	      <office>216</office>
	   </employee>
	</dept>

WWPR	<dept deptID="V15">
	   <employee id="673">
	      <name>Matt Foreman</name>
	      <phone>416 891 7301</phone>
	      <office>216</office>
	   </employee>
	   <description>This dept supports sales world wide</description>
	</dept>

S-USE	...
...	...

Figure 27. Sample data for the DEPT table

GUPI

This intermediate granularity is a reasonable choice if a department is the predominant granularity at which your application accesses and processes the data. Alternatively, you might decide to combine multiple departments into a single XML document, such as those that belong to one unit. This coarse granularity, however, is sub-optimal if your application typically processes only one department at a time.

You might also choose one XML document per employee with an additional "dept" attribute for each employee to indicate which department he or she belongs to. This fine granularity would be a very good choice if employees use business objects of interest, which are often accessed and processed independently from the other employees in the same department. However, if the application typically processes all employees in one department together, one XML document per department might be the better choice.

Use attributes and elements appropriately in XML

A common question related to XML document design, is when to use attributes instead of elements, and how that choice affects performance.

This question is much more relevant to data modeling than to performance. However, as a general rule, XML elements are more flexible than attributes because they can be repeated and nested.

For example, the department documents shown in the preceding example, use an element "phone" which allows multiple occurrences of "phone" for an employee who has multiple numbers. This design is also extensible in case we later need to break phone numbers into fragments, such as child elements for country code, area code, extension, and so on.

By contrast, if "phone" is an attribute of the employee element instead, it can exist only once per employee, and you could not add child elements. Such limitations might hinder future schema evolution.

Although you can probably model all of your data without using attributes, they can be a very intuitive choice for data items that are known not to repeat for a single element, nor have any sub-fields. Attributes can reduce the size of XML data slightly because they have only a single name-value pair, as opposed to elements, which have both a start tag and an end tag.

In DB2, you can use attributes in queries, predicates, and index definitions just as easily as elements. Because attributes are less extensible than elements, DB2 can apply certain storage and access optimizations. However, these advantages should be considered an extra performance bonus rather than an incentive to convert elements to attributes for the sake of performance, especially when data modeling considerations call for elements.

Be aware of XML schema validation overhead

XML schema validation is an optional activity during XML parsing. Performance studies have shown that XML parsing in general is significantly more CPU-intensive if schema validation is enabled.

This overhead can vary drastically depending on the structure and size of your XML documents, and particularly on the size and complexity of the XML Schema used. For example, you might find 50% higher CPU consumption because of schema validation with moderately complex schemas. Unless your XML inserts are heavily I/O bound, the increased CPU consumption typically translates to reduced insert throughput.

An XML schema defines the structure, elements and attributes, data types, and value ranges, that are allowed in a set of XML documents. DB2 allows you to validate XML documents against XML schemas. If you choose to validate documents, you typically do so at insert time. Validation ensures that data inserted into the database is compliant with the schema definition, meaning that you improve the integrity of data entering your tables.

Consider the impact to performance, when you determine whether your application needs the stricter type checking for XML queries and XML schema compliance. For example, if you are using an application server which receives, validates, and processes XML documents before they are stored in the database, the documents probably do not need to be validated again in DB2. At that point you already know they are valid. Likewise, if the database receives XML documents from a trusted application, maybe even one that you control, and you know that the XML data is always valid, avoid schema validation for the benefit of higher insert performance. If, however, your DB2 database receives XML data from untrusted sources and you need to ensure schema compliance at the DB2 level, then you need to spend some extra CPU cycles on that.

Related information:

XML schema validation (DB2 Programming for XML)

Prerequisites for using pureXML (DB2 Programming for XML)

Specify full paths in XPath expressions when possible

When you know where in the structure of an XML document the element is located, provide that information in the form of a fully specified path to avoid unneeded overhead.

GUPI

Consider the table that is created by the following SQL statement.

```
CREATE TABLE customer(info XML);
```

The following figure shows sample data in the info column.

```
<customerinfo Cid="1004">
    <name>Matt Foreman</name>
    <addr country="Canada">
          <street>1596 Baseline</street>
          <city>Toronto</city>
          <state/>Ontario
          <pcode>M3Z-5H9</pcode>
    </addr>
    <phone type="work">905-555-4789</phone>
    <phone type="home">416-555-3376</phone>
</customerinfo>
```

Figure 28. Sample data in a customerinfo XML document

If you want to retrieve customer phone numbers or the cities where they live, you can choose from several possible path expressions to get that data.

Both `/customerinfo/phone` and `//phone` would get you the phone numbers. Likewise, `/customerinfo/addr/city` and `/customerinfo/*/city` both return the city. For best performance, the fully specified path is preferred over using either `*` or `//` because the fully specified path enables DB2 to navigate directly to the elements, skipping over non-relevant parts of the document. If you ask for `//phone` instead of /customerinfo/phone, you ask for phone elements anywhere in the document. This requires DB2 to navigate down into the "addr" subtree of the document to look for phone elements at any level of the document.

Using * and // also can lead to unexpected query results (for example, if some of the "customerinfo" documents also contain "assistant" information, as shown in the following figure). The path //phone would return the customer phones and the assistant phone numbers, without distinguishing them. From the query result, you might mistakenly process the assistant's phone as a customer phone number.

```
<customerinfo Cid="1004">
    <name>Matt Foreman</name>
    <addr country="Canada">
          <street>1596 Baseline</street>
          <city>Toronto</city>
          <state/>Ontario
          <pcode>M3Z-5H9</pcode>
    </addr>
    <phone type="work">905-555-4789</phone>
    <phone type="home">416-555-3376</phone>
    <assistant>
          <name>Peter Smith</name>
          <phone type="home">416-555-3426</phone>
     </assistant>
</customerinfo>
```

Figure 29. Sample data with phone and name elements at multiple levels

GUPI

Define lean indexes for XML data to avoid extra overhead

Assume that queries often search for "customerinfo" XML documents by customer name. An index on the customer name element, as shown in the following statements, can greatly improve the performance of such queries.

GUPI

```
CREATE TABLE CUSTOMER (info XML);

CREATE INDEX custname1 ON customer(info)
GENERATE KEY USING XMLPATTERN '/customerinfo/name' as sql varchar(20);

CREATE INDEX custname2 ON customer(info)
GENERATE KEY USING XMLPATTERN '//name' as sql varchar(20);

SELECT * FROM customer
WHERE XMLEXISTS('$i/customerinfo[name = "Matt Foreman"]' passing info as $i);
```

Both of the indexes defined above are eligible to evaluate the XMLEXISTS predicate on the customer name. However, the custname2 index might be substantially larger than the custname1 index because it contains index entries not only for customer names but also for assistant names. This is because the XML pattern //name matches name elements anywhere in the document. However, if we never search by assistant name then we don't need them indexed.

For read operations, the custname1 index is smaller and therefore potentially better performing. For insert, update and delete operations, the custname1 index incurs index maintenance overhead only for customer names, while the custname2 index requires index maintenance for customer and assistant names. You certainly don't want to pay that extra price if you require maximum insert, update, and delete performance and you don't need indexed access based on assistant names.

GUPI

Related information:

Storage structure for XML data (Introduction to DB2 for z/OS)

XML data indexing (DB2 Programming for XML)

Access methods with XML indexes (DB2 Programming for XML)

Use XMLEXISTS for predicates that filter at the document level

GUPI

Consider the following table and sample data:

```
CREATE TABLE customer(info XML);
```

```
<customerinfo>
    <name>Matt Foreman</name>
    <phone>905-555-4789</phone>
</customerinfo>

<customerinfo>
    <name>Peter Jones</name>
    <phone>905-123-9065</phone>
</customerinfo>

<customerinfo>
    <name>Mary Clark</name>
    <phone>905-890-0763</phone>
</customerinfo>
```

Figure 30. Sample data in the customer table

Assume, for example, that you want to return the names of customers which have the phone number "905-555-4789". You might be tempted to write the following query.

```
SELECT XMLQUERY('$i/customerinfo[phone = "905-555-4789"]/name' passing info as "i")
FROM customer;
```

However, this query is not what you want for several reasons:

- It returns the following result set which has as many rows as there are rows in the table. This is because the SQL statement has no where clause and therefore cannot eliminate any rows. The result is shown in the following figure.

  ```
  <name>Matt Foreman</name>

  3 record(s) selected
  ```

Figure 31. Result for the preceding example query

- For each row in the table which doesn't match the predicate, a row containing an empty XML sequence is returned. This is because the XQuery expression in the XMLQUERY function is applied to one row, or document, at a time and never removes a row from the result set, only modifies its value. The value produced by that XQuery is either the customer's name element if the predicate is true, or the empty sequence otherwise. These empty rows are semantically correct, according to the SQL/XML standard, and must be returned if the query is written as shown.
- The performance of the query is poor. First, an index which might exist on /customerinfo/phone cannot be used because this query is not allowed to eliminate any rows. Secondly, returning many empty rows makes this query needlessly slow.

To resolve the performance issues and get the output that you want, use the XMLQUERY function in the select clause only to extract the customer names, and move the search condition, which should eliminate rows, into an XMLEXISTS predicate in the WHERE clause. Doing so will allow index usage, row filtering, and avoid the overhead of empty results rows. You could write the query as shown in the following figure.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE XMLEXISTS('$i/customerinfo[phone = "905-555-4789"]' passing info as "i")
```

GUPI

Related information:

XMLEXISTS predicate for querying XML data (DB2 Programming for XML)

XMLEXISTS predicate (DB2 SQL)

Use square brackets to avoid Boolean predicates in XMLEXISTS

GUPI

A common error is to write the previous query without the square brackets in the XMLEXISTS function, as shown in the following query.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE XMLEXISTS('$i/customerinfo/phone = "905-555-4789"' passing info as "i")
```

Writing the query this way produces the following results shown in the following figure.

```
<name>Matt Foreman</name>
<name>Peter Jones</name>
<name>Mary Clark</name>

3 record(s) selected
```

Figure 32. Sample results for the preceding example query

The expression in the XMLEXISTS predicate is written such that XMLEXISTS always evaluates to true. Hence, no rows are eliminated. For a given row, the XMLEXISTS predicate evaluates to false only if the XQuery expression inside returns the empty sequence. However, without the square brackets the XQuery expression is a Boolean expression which always returns a Boolean value and never the empty sequence. Note that XMLEXISTS truly checks for the existence of a value and evaluates to true if a value exists, even if that value happens to be the Boolean value "false". This behavior is correct according to the SQL/XML standard, although it is probably not what you intended to express.

The impact is again that an index on phone cannot be used because no rows will be eliminated, and you receive a lot more rows than you actually want. Also, beware not to make this same mistake when using two or more predicates, as shown in the following query.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE XMLEXISTS('$i/customerinfo[phone = "905-555-4789"] and
   $i/customerinfo[name = "Matt Foreman"]'
      passing info as "i")
```

The XQuery expression is still a Boolean expression because it has the form "exp1 and exp2." You would write the query as shown in the following query to filter rows and allow for index usage.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
from customer
WHERE XMLEXISTS('$i/customerinfo[phone = "905-555-4789" and name = "Matt Foreman"]'
      passing info as "i")
```

GUPI

Related information:

Logical expressions (DB2 Programming for XML)

Use RUNSTATS to collect statistics for XML data and indexes

The RUNSTATS utility has been extended to collect statistics on XML tables and indexes, and DB2 optimizer uses these statistics to generate efficient execution plan for XML queries. Consequently, continue to use RUNSTATS on XML tables and indexes as you would for relational data. You need to specify XML table space names explicitly or use LISTDEF to include ALL or XML objects to obtain the XML table statistics.

Related information:

Collection of statistics on XML objects (DB2 Utilities)

RUNSTATS (DB2 Utilities)

Use SQL/XML publishing views to expose relational data as XML

GUPI

You can include relational columns in a SQL/XML publishing view, and when querying the view, express any predicates on those columns rather than on the constructed XML.

SQL/XML publishing functions allow you to convert relational data into XML format. Hiding the SQL/XML publishing functions in a view definition can be beneficial. Applications or other queries can simply select the constructed XML documents from the view, instead of dealing with the publishing functions themselves. The following statements creates a view that contains hidden SQL/XML publishing functions.

```
CREATE TABLE unit( unitID char(8), name char(20), manager varchar(20));

CREATE VIEW UnitView(unitID, name, unitdoc) as
   SELECT unitID, name,
          XMLELEMENT(NAME "Unit",
             XMLELEMENT(NAME "ID", u,unitID),
             XMLELEMENT(NAME "UnitName", u.name),
             XMLELEMENT(NAME "Mgr", u.manager)
                  )
   FROM unit u;
```

Note that the view definition includes relational columns. This does not create any physical redundancy because it is only a view, not a materialized view. Exposing the relational columns helps to query this view efficiently.

The following query uses a relational predicate to ensure that only the XML document for "WWPR" is constructed, resulting in a shorter run time, especially on a large data set.

```
SELECT unitdoc
FROM UnitView
WHERE unitID = "WWPR";
```

GUPI

Use XMLTABLE views to expose XML data in relational format

GUPI

You might also want to use a view to expose XML data in relational format. Similar caution needs to be applied as before, but in the reverse way. In the following example the SQL/XML function XMLTABLE returns values from XML documents in tabular format.

```
CREATE TABLE customer(info XML);

CREATE VIEW myview(CustomerID, Name, Zip, Info) AS
SELECT T.*, info
FROM customer, XMLTABLE ('$c/customerinfo' passing info as "c"
   COLUMNS
   "CID"     INTEGER      PATH './@Cid',
   "Name"    VARCHAR(30)  PATH './name',
   "Zip"     CHAR(12)     PATH './addr/pcode' ) as T;
```

The view definition includes XML column info to help query the view efficiently. Assume that you want to retrieve a tabular list of customer IDs and names for a given ZIP code. Both of the following queries can do that, but the second one tends to perform better than the first.

In the first query, the filtering predicate is expressed on the CHAR column "Zip" generated by the XMLTABLE function. However, not all the relational predicates can be applied to the underlying XML column or indexes. Consequently, the query requires the view to generate rows for all customers and then picks out the one for zip code "95141".

```
SELECT CustomerID, Name
FROM myview
WHERE Zip = '95141';
```

The second query uses an XML predicate to ensure that only the rows for "95141" get generated, resulting in a shorter run time, especially on a large data set.

```
SELECT CustomerID, Name
FROM myView
WHERE xmlexists('$i/customerinfo[addr/pcode = "95141"]' passing info as "i");
```

GUPI

Use SQL and XML statements with parameter markers for short queries and OLTP applications

GUPI

The SQL/XML functions XMLQUERY, XMLTABLE and XMLEXISTS support external parameters.

Very short database queries often execute so fast that the time to compile and optimize them is a substantial portion of their total response time. Consequently, you might want to compile, or "prepare," them just once and only pass predicate literal values for each execution. This technique is recommended for applications with short and repetitive queries. The following query shows how you can use parameter markers to achieve the result of the preceding example.

```
SELECT info
FROM customer
WHERE xmlexists('$i/customerinfo[phone = $p]'
                passing info as "i", cast(? as varchar(12)) as "p")
```

GUPI

Avoid code page conversion during XML insert and retrieval

GUPI

XML is different from other types of data in DB2 because it can be internally and externally encoded. Internally encoded means that the encoding of your XML data can be derived from the data itself. Externally encoded means that the encoding is derived from external information.

The data type of the application variables that you use to exchange XML data with DB2 determines how the encoding is derived. If your application uses character type variables for XML, then it is externally encoded. If you use binary application data types, then the XML data is considered internally encoded.

Internally encoded means that the encoding is determined by either a Unicode Byte-Order mark (BOM) or an encoding declaration in the XML document itself, such as: `<?xml version="1.0" encoding="UTF-8" ?>`

From a performance point of view, the goal is to avoid code page conversions as much as possible because they consume extra CPU cycles. Internally encoded XML data is preferred over externally encoded data because it can prevent unnecessary code page conversion.

This means that in your application you should prefer binary data types over character types. For example, in ODBC when you use `SQLBindParameter()` to bind parameter markers to input data buffers, you should use SQL_C_BINARY data buffers rather than `SQL_C_CHAR`, `SQL_C_DBCHAR`, or `SQL_C_WCHAR`. In host applications, use XML AS BLOB as the host variable type.

When inserting XML data from Java applications, reading in the XML data as a binary stream (`setBinaryStream`) is better than as a string (`setString`). Similarly, if your Java application receives XML from DB2 and writes it to a file, code page conversion may occur if the XML is written as non-binary data.

When you retrieve XML data from DB2 into your application, it is serialized. Serialization is the inverse operation of XML parsing. It is the process that DB2 uses to convert internal XML format, which is a parsed, tree-like representation, into the textual XML format that your application can understand. In most cases it is best to let DB2 perform implicit serialization. This means your SQL/XML statements simply select XML-type values as shown in the following example, and that DB2 performs the serialization into your application variables as efficiently as possible.

```
CREATE TABLE customer(info XML);

SELECT info FROM customer WHERE...;

SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE...;
```

If your application deals with very large XML documents, it might benefit from using LOB locators for data retrieval. This requires explicit serialization to a LOB type, preferably BLOB, because explicit serialization into a character type such as CLOB can introduce encoding issues and unnecessary code page conversion. Explicit serialization uses the XMLSERIALIZE function as shown in the following query.

```
SELECT XMLSERIALIZE(info as BLOB(1M)) FROM customer WHERE...;
```

GUPI

Use the XMLMODIFY statement to update part of an XML document

GUPI

When you need to modify only part of an XML document, you can use the XMLMODIFY function to make the changes to the XML data more efficiently than by replacing the entire XML document, especially in cases of large XML documents. For small XML documents, no performance advantage is provided by the XMLMODIFY statement for documents that fit within a single record.

When an application does not use the XMLMODIFY statement to update an XML column, the XML document from the XML column is entirely deleted and replaced by a new XML document. When the XMLMODIFY is used to update an XML column, only the rows in the XML table space that are modified by the XMLMODIFY function need to be deleted or replaced.

GUPI

Related information:

Partial updates of XML documents (DB2 Programming for XML)

XMLMODIFY (DB2 SQL)

Use the Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format for input data for parsed XML documents

The parsing of XML data is one of the most significant factors that affect performance during INSERT, LOAD, and UPDATE operations for XML data. If you use Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format when you insert, update, or load data, CPU overhead is reduced. DB2 DRDA zIIP redirect is not affected by binary XML, but z/OS XML System Services zIIP and zAAP are affected by binary XML because parsing is not needed.

Sending Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format data from a Java application on a client reduces the CPU time needed on the DB2 server. However, parsing the XML data into binary form is handled by the IBM Data Server Driver for JDBC and SQLJ running on the client. Class 1 elapsed time on the client might increase when compared to sending textual XML. Use binary XML to reduce CPU time on the DB2 server if the increase of elapsed time does not impact your environment.

Consider performance when you decide when to use XPath or XQuery

In general, if you perform similar operations using XQuery and XPath, your performance should be similar. However, in some situations, XPath might provide better performance.

Related information:

Best applications for XQuery or XPath (DB2 Programming for XML)

Overview of XQuery (DB2 Programming for XML)

Selecting XML data (DB2 Application programming and SQL)

Set the XML_RANDOMIZE_DOCID subsystem parameter for best performance.

You can reduce wait times for inserting XML data by randomizing DOCID values for tables that contain XML columns. When the DOCID values are inserted in sequential order, hot spot situations might occur, in which multiple threads must wait for latches on the same data pages while inserting XML data concurrently.

However, when the value of the XML_RANDOMIZE_DOCID subsystem parameter is YES, DB2 randomizes the DOCID values at CREATE TABLE for any new table that contains XML columns, and at ALTER TABLE for any existing table when the first XML column is added. Changing the value of the XML_RANDOMIZE_DOCID subsystem parameter has no effect on existing tables that contain XML columns. Any table that contains randomized DOCID values cannot be converted to use sequential DOCID values. Similarly, any table that already contains sequential DOCID values cannot be converted to use randomized DOCID values.

You can check the value of the ORDER column in the SYSIBM.SYSSEQUENCES catalog table to find out whether a particular table has randomized DOCID values.

Related information:

RANDOMIZE XML DOCID field (XML_RANDOMIZE_DOCID subsystem parameter) (DB2 Installation and Migration)

SYSIBM.SYSSEQUENCES table (DB2 SQL)

Access XML data quickly by using FETCH WITH CONTINUE

GUPI

Use the FETCH WITH CONTINUE statement to improve the performance of some queries that reference XML columns with unknown or very large maximum lengths.

GUPI

Related tasks:

Accessing XML or LOB data quickly by using FETCH WITH CONTINUE (DB2 Application programming and SQL)

Part 7. Maintaining data organization and statistics

You can enable DB2 to choose the most efficient access paths by reorganizing your data and collecting accurate statistics.

About this task

DB2 uses the catalog statistics in conjunction with information about database design, and the details of the SQL statement to choose access paths.

Space usage statistics also help DB2 to select access paths that use index or table space access as efficiently as possible. By reducing gaps between leaf pages in an index, or by ensuring that data pages are well-organized, you can reduce the use of inefficient I/O operations.

Procedure

To ensure that the statistics in the DB2 catalog accurately reflect the organization and content of your data:

1. Invoke the REORG utility to reorganize the necessary tables, including the DB2 catalog table spaces and user table spaces. You can invoke the DSNACCOX stored procedure to determine when reorganization is needed.
2. Invoke the RUNSTATS utility to capture statistics.
3. Rebind the plans or packages that contain affected queries. Specify the PLANMGMT bind option to save previous copies of the packages. You can use the APCOMPARE bind option to detect access path changes for you static SQL statements. For dynamic SQL statements, DB2 uses the newly collected statistics at the next prepare.
4. Capture EXPLAIN information to validate access path changes.
5. In the event of access path regression, use the REBIND command and specify the SWITCH option to revert to a previous access path. This action depends upon the PLANMGMT bind option that was specified when packages were first rebound.

What to do next

Implement a strategy for reorganizing your data and collecting statistics routinely. Routine statistics collection is necessary for maintaining good performance, and is likely to be required at additional times, other than after reorganization.You can also use RUNSTATS profiles and certain stored procedures to automate statistics maintenance.

PSPI

Related tasks:

Investigating access path problems

Related reference:

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

RUNSTATS (DB2 Utilities)

DSNACCOX stored procedure (DB2 SQL)

Chapter 33. Maintaining data organization

Data that is physically well-organized can improve the performance of access paths that rely on index or table scans, and reduce the amount of disk storage used for the data.

About this task

Depending on the number of changes, you might encounter performance degradations for the following types of operations when your data becomes disorganized:

- Dynamic SQL queries.
- Updates and deletes. For example, delete operations sometimes result in pseudo-deleted index entries, which can result in additional lock contention.
- ALTER statements (especially those that run concurrently).
- Concurrent REORG and LOAD utilities.
- Unloading a table that has had many changes before reorganization.

Procedure

To determine when to reorganize your data, use any of the following approaches:

- Monitor statistics for increases in the following values:
  - I/O operations
  - Get page operations
  - Processor consumption

  When performance degrades to an unacceptable level, analyze the statistics to develop your own rules for when to reorganize the data in your particular environment.
- Invoke the DSNACCOX stored procedure to get recommendations based on real-time statistics values.
- Query the catalog. Member DSNTESP of the SDSNSAMP data set contains sample useful queries for determining whether to reorganize
- Use the REORG utility. The REORG utility embeds the function of catalog queries. If a query returns a certain result (you can use the default or supply your own), REORG either reorganizes or does not reorganize. Optionally, you can have REORG run a report instead of actually doing the reorganization. The following REORG options invoke the catalog queries:
  - The OFFPOSLIMIT and INDREFLIMIT options of REORG TABLESPACE
  - The LEAFDISTLIMIT option of REORG INDEX
- Always reorganize your data after table definitions change. When ALTER TABLE statements are used to make any to the following changes to table the table space is placed in advisory REORG-pending (AREO*) status:
  - Add columns
  - Data type changes
  - Changed column lengths

  When an existing column is changed, the table space is placed in AREO* status because the conversion to the new definition is not immediate. Reorganizing the table space causes the rows to be reloaded with the data converted to the new definition. Until the table space is reorganized, the changes must be tracked and applied as the data is accessed, which might degrade performance.

Related concepts:

Objects that are subject to locks

Related reference:

DB2 catalog tables (DB2 SQL)

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

DSNACCOX stored procedure (DB2 SQL)

Related information:

DB2 for z/OS and List Prefetch Optimizer (IBM Redbooks)

Determining when to reorganize indexes

You can use data in the SYSIBM.SYSINDEXSPACESTATS table to determine when to reorganize an index.

Before you begin

For best results, consider calling the DSNACCOX stored procedure to determine whether to reorganize database objects.

Procedure

To investigate whether to reorganize indexes:

- Reorganize the indexes if any of the following conditions are true in the SYSIBM.SYSINDEXSPACESTATS catalog table:
  - REORGPSEUDODELETES/FLOAT(TOTALENTRIES) > 10% in a non-data sharing environment, or REORGPSEUDODELETES/TOTALENTRIES > 5% in a data sharing environment.
  - REORGINSERTS/FLOAT(TOTALENTRIES) > 25%
  - REORGDELETES/FLOAT(TOTALENTRIES) > 25%
  - REORGAPPENDINSERT/FLOAT(TOTALENTRIES) > 20%
  - EXTENTS > 254
- Reorganize the indexes if any of the following conditions are true:
  - Advisory REORG-pending state (AREO*) as a result of an ALTER statement.
  - Advisory REBUILD-pending state (ARBDP) as a result an ALTER statement.

Related tasks:

Reorganizing indexes (DB2 Administration Guide)

Using the LEAFDISTLIMIT and REPORTONLY options to determine when reorganization is needed (DB2 Utilities)

Related reference:

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

REORG-pending status (DB2 Utilities)

REBUILD-pending status (DB2 Utilities)

REORG INDEX (DB2 Utilities)

LEAFNEAR and LEAFFAR columns

The LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART measure the disorganization of physical leaf pages by indicating the number of pages that are not in an optimal position.

Leaf pages can have page gaps whenever index pages are deleted or when an insert that cannot fit onto a full page causes an index leaf page that cannot fit onto a full page. If the key cannot fit on the page, DB2 moves half the index entries onto a new page, which might be far away from the "home" page.

Figure 33 shows the logical and physical view of an index.

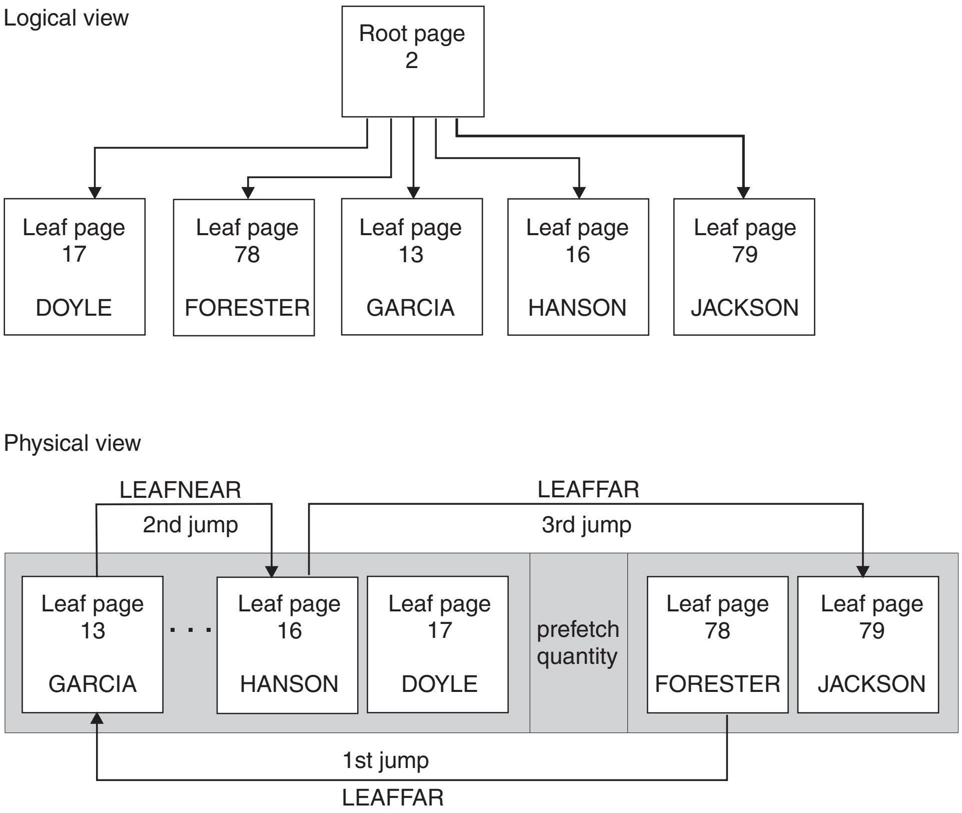


Figure 33. Logical and physical views of an index in which LEAFNEAR=1 and LEAFFAR=2

The logical view at the top of the figure shows that for an index scan four leaf pages need to be scanned to access the data for FORESTER through JACKSON. The physical view at the bottom of the figure shows how the pages are physically accessed. The first page is at physical leaf page 78, and the other leaf pages are at physical locations 79, 13, and 16. A jump forward or backward of more than one page represents non-optimal physical ordering. LEAFNEAR represents the number of jumps within the prefetch quantity, and LEAFFAR represents the number of jumps outside the prefetch quantity. In this example, assuming that the prefetch quantity is 32, two jumps exist outside the prefetch quantity. A jump from page 78 to page 13, and one from page 16 to page 79. Thus, LEAFFAR is 2. Because of the jump within the prefetch quantity from page 13 to page 16, LEAFNEAR is 1.

LEAFNEAR has a smaller impact than LEAFFAR because the LEAFNEAR pages, which are located within the prefetch quantity, are typically read by prefetch without incurring extra I/Os.

The optimal value of the LEAFNEAR and LEAFFAR catalog columns is zero. However, immediately after you run the REORG and gather statistics, LEAFNEAR for a large index might be greater than zero. A non-zero value could be caused by free pages that result from the FREEPAGE option on CREATE INDEX, non-leaf pages, or various system pages; the jumps over these pages are included in LEAFNEAR.

Deciding when to reorganize table spaces

You can used values in the SYSIBM.SYSTABLESPACESTATS catalog table determine when to reorganize a table space.

Before you begin

For best results, consider calling the DSNACCOX stored procedure to determine whether to reorganize database objects.

Procedure

- Reorganize table spaces when any of the following conditions are true. These conditions are based on values in the SYSIBM.SYSTABLESPACESTATS table.
  - REORGUNCLUSTINS/FLOAT(TOTALROWS) > 10%

    Do not use REORGUNCLUSTINS to determine if you should run REORG if access to the table space is predominantly random access.
  - (REORGNEARINDREF+REORGFARINDREF)/FLOAT(TOTALROWS) > 5% in a data sharing environment, or (REORGNEARINDREF+REORGFARINDREF)/FLOAT(TOTALROWS) >10% in non-data sharing environment
  - REORGINSERTS/FLOAT(TOTALROWS) > 25%
  - REORGDELETES/FLOAT(TOTALROWS) > 25%
  - EXTENTS > 254
  - REORGDISORGLOB/FLOAT(TOTALROWS) > 50%
  - SPACE > 2 * (DATASIZE / 1024)
  - REORGMASSDELETE > 0
  - REORGCLUSTSENS > 0
- Reorganize table spaces if any of the following conditions are true:
  - The table space is in the advisory REORG-pending state (AREO*) as a result of an ALTER TABLE statement.
  - An index on a table in the table space is in the advisory REBUILD-pending state (ARBDP) as result an ALTER TABLE statement.

Related reference:

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

REORG TABLESPACE (DB2 Utilities)

Reorganizing LOB table spaces

You can reorganize the data in LOB table spaces to reclaim disk space.

About this task

SYSIBM.SYSLOBSTATS contains information about how the data in the table space is physically stored.

Procedure

To determine when to reorganize LOB table spaces:

- Consider running REORG on the LOB table space when the percentage in the ORGRATIO column is less than 80.
- Use real-time statistics to identify DB2 objects that should be reorganized, have their statistics updated, or be image-copied.

Chapter 34. Maintaining DB2 database statistics

You can improve the ability of DB2 to choose efficient access paths by collecting database statistics routinely to keep the statistics accurate, up to date, and in sync for related objects.

About this task

The RUNSTATS utility collects statistics for database objects. You can also use *inline statistics* to collect statistics for database objects, when you run other utilities, such as the LOAD, REBUILD INDEX, REORG INDEX, and REORG TABLESPACES.

DB2 stores the statistics in catalog tables.DB2 uses the statistics during the bind process to determine the most efficient access paths. So, if you never collect statistics and later rebind your packages or plans, DB2 cannot have the information that it needs to choose the most efficient access path. The result might be unnecessary I/O operations and excessive processor consumption.

Tip: You can reduce the effort that is required to maintain your DB2 database statistics by using statistics profiles and automating the collection of statistics. For details, see Automating statistics maintenance and Statistics profiles.

Procedure

Use the following approaches to maintain database statistics:

- Collect statistics at least once against each table and all of its associated indexes. You might not need to collect statistics again for objects that are stable in size and contain data values that change infrequently.
- Whenever you collect statistics for a table or index, always collect statistics for the entire table space and all associated indexes. If you collect statistics for a single partition, collect statistics at the table space level and index statistics for that partition. The goal is to maintain consistent statistics for related objects so that DB2 has consistent information for access path selection. When the statistics for related objects are out of sync, DB2 cannot discern between the accurate and inaccurate statistics, and suboptimal access paths can result.
- Collect statistics routinely for tables that have characteristics that vary over time. For example, you might prefer any of several common approaches to routine statistics collection:
  - Collect statistics on a routine schedule or based on data volume growth, regardless of when data is reorganized.
  - Collect statistics only after you reorganize data the table space level. The goal of this approach is to maintain perfect cluster ratio statistics. If you follow this approach, do not collect statistics on a single index after you reorganize it. Instead, wait until the next table space level reorganization.
  - Use DSNACCOX stored procedure to get recommendations for when to collect statistics.
  - Collect statistics only when data or indexes change significantly, such as in the following situations:

    - When you load data into a table, and before you bind application plans and packages that access the table. You can collect statistics inline with the LOAD utility.
    - After you create an index. Before a statically bound application can use a new index, you must rebind the application plan or package. Dynamic SQL statements can take advantage of a new index at the next prepare for each statement.
    - After certain ALTER statements, including any that requires you to drop and re-create an object, ALTER TABLE ROTATE PARTITION, or ALTER TABLE ALTER COLUMN that changes the data type, length (except for VARCHAR columns), precision, scale, or subtype of the column .
    - After utilities such as RECOVER or REBUILD, and especially point-in-time recovery.
    - After a REORG utility operation that resolves pending definition changes from certain types of ALTER statements.
    - After heavy insert, update, and delete activity.

- Tables that constantly change size present difficulties for the collection of useful statistics. Such tables might not warrant frequent statistics collection because of the difficulty of collecting statistics at a representative time. You can use the following approaches for such tables:
  - Define tables as VOLATILE, or specify a value for the NPGTHRSH subsystem parameter, to favor index access whenever possible. If statistics indicate that such tables are empty during bind or prepare, DB2 can sometimes use real-time statistics to find size information for such tables. However, real-time statistics do not contain cardinality information and they cannot replace the collection of complete statistics in all cases.
  - Attempt to collect statistics when the table contains some representative data, regardless of whether you can determine exact peak amount.
- Collect statistics periodically for DB2 catalog objects to provide DB2 with more accurate information for access path selection for user queries to the catalog.
- Use care when you use SQL statements or tools to update statistics. If such updates introduce invalid data, they can cause unpredictable results, including abends for RUNSTATS and other utilities. If such problems occur, you can run the RUNSTATS utility and collect statistics at the table space level to resolve the problems, in most cases.

What to do next

- Decide whether you need to rebind your packages after you collect statistics.
- Monitor catalog statistics with EXPLAIN data to ensure that your queries access data efficiently.

Related concepts:

Access path selection in a data sharing group (DB2 Data Sharing Planning and Administration)

Related reference:

RUNSTATS (DB2 Utilities)

NPGTHRSH in macro DSN6SPRM (DB2 Installation and Migration)

DSNACCOX stored procedure (DB2 SQL)

Collecting statistics by using DB2 utilities

You can run certain DB2 utilities to collect access path statistics for your database objects. Accurate statistics are an essential component of access path selection.

Before you begin

- Consider identifying statistics to collect by using a query optimization tool such as IBM Data Studio or IBM Data Server Manager.

About this task

You can use DB2 online utilities to collect statistics for database objects. The purpose of the RUNSTATS utility is to collect statistics for database objects. However, you can also collect *inline statistics* when you run certain other DB2 utilities.

Procedure

To collect statistics for database objects, use any of the following approaches:

- Run the RUNSTATS utility. The RUNSTATS utility collects the most complete and accurate statistics.
- Specify the STATISTICS keyword to specify the collection of inline statistics when you run one of the following utilities:
  - LOAD
  - REBUILD INDEX
  - REORG INDEX
  - REORG TABLESPACE

  You might be able to avoid the cost of running the RUNSTATS utility by collecting inline statistics.

  However, certain limitations apply to inline statistics. For example:

  - You cannot collect column group statistics with the STATISTICS keyword. You must run the RUNSTATS utility to collect column group statistics.
  - You cannot use statistics profiles with inline statistics.

  For details of the limitations, see the information for each utility.

  Important: Statistics that are collected with inline statistics are likely to differ from statistics that are collected by the RUNSTATS utility. Certain resources and values that RUNSTATS uses might be unavailable in the context of a utility that collects inline statistics. Estimations must be used in place of these missing values or uncertainties, and the resulting statistics might be less exact. Consequently, you might need to evaluate whether the inline statistics are suitable to support access path selection for your query workload.

  For example:

  - If the DISCARDDN option is specified when you collect inline statistics with the LOAD utility, the statistics are collected before the rows are discarded. If the number of discarded rows is large enough, the inaccuracy of the resulting statistics might be significant. As a general rule, if the number of discarded rows exceeds 20 percent of the total number of rows in the table, run the RUNSTATS separately, after running the LOAD utility, to collect accurate statistics.

What to do next

Consider taking the following actions to standardize and automate statistics collection:

- Consider whether to rebind application programs that depend on the statistics values for access path selection.
- Create statistics profiles to standardize statistics collection options. Only the RUNTATS utility supports statistics profiles.
- Automate statistics maintenance.

Related tasks:

Improving filter factors by collecting cardinality and frequency statistics

Reducing the cost of collecting statistics

Related reference:

RUNSTATS (DB2 Utilities)

LOAD (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

Improving filter factors by collecting cardinality and frequency statistics

You can improve the filter factors that DB2 uses for access path selection by collecting cardinality and frequency statistics for columns and columns groups that are used in predicates.

Before you begin

- Consider identifying appropriate statistics to collect for your query workload by using a query optimization tool.

About this task

DB2 needs an accurate estimate of the number of rows that qualify each predicate is applied to determine optimal access paths. When multiple tables are accessed, filtering also affects the join order, the join method, and the cost of the join.

The SYSIBM.SYSCOLUMNS and SYSIBM.SYSCOLDIST catalog tables are the main source of statistics for calculating predicate filter factors.

Cardinality statistics

The COLCARDF column of the SYSCOLUMNS catalog table indicates the *cardinality* of a column. A positive value is an estimate of the number of distinct values in the column. When no statistics are collected for the column, the value is -1, and a default filter factor might be used.

The value of the COLCARDF column that the RUNSTATS TABLESPACE utility generates is an estimate that is determined by a sampling method. If you know a more accurate value, you can supply it by updating the catalog. If the column is the first column of a non-DPSI index, the value that the RUNSTATS INDEX utility generates is exact.

Frequency or distribution statistics

Columns in the SYSCOLDIST catalog table contain *frequency statistics* (sometimes also called *distribution statistics*) for the values in a single column.

When frequency statistics do not exist, DB2 assumes that the data is uniformly distributed and all values in the column occur with the same frequency. This assumption can lead to an inaccurate estimate of the number of qualifying rows if the data is skewed, which can result in performance problems.

For example, assume that a column (AGE_CATEGORY) contains five distinct values (COLCARDF), each of which occur with the following frequencies:

```
AGE_CATEGORY          FREQUENCY
------------          ---------
INFANT                  5%
CHILD                  15%
ADOLESCENT             25%
ADULT                  40%
SENIOR                 15%
```

Without this frequency information, DB2 must use a default filter factor of 1/5 (or 1/COLCARDF), or 20%, to estimate the number of rows that qualify for predicate AGE_CATEGORY=ADULT. However, the actual frequency of that age category is 40%. Thus, the number of qualifying rows is underestimated by 50%.

Procedure

To improve predicate filter factors by collecting frequency and statistics, use any of the following approaches:

- When you collect statistics at the table level, collect statistics only for columns or column groups that might be used as search conditions in WHERE clauses of queries.
  - Specify the COLUMN option to collect statistics for specified columns only.
  - Specify the COLGROUP option to collect statistics for specified groups of columns only. You can also specify the FREQVAL option with the COLGROUP option to collect distribution statistics for the column group. You cannot specify the COLGROUP option when you collect inline statistics by using utilities other than RUNSTATS.

  When you collect statistics on groups of columns that are used in predicates, the improved accuracy of the filter factor estimate can lead to improved query performance. However, collecting statistics on all columns of a table is costly and might be unnecessary.
- Collect cardinality statistics on all columns that are used as predicates in WHERE clauses.
- Collect frequency statistics for all columns that have low cardinality values and are used in `COL` *`op`* *`constant`* predicates.
- Collect frequency statistics for columns that can contain default data if the default data is skewed and the column is used in a `COL` *`op`* *`constant`* predicate.
- Collect column group statistics on all columns that are used in join predicates.
- Collect column statistics periodically for columns that contain data with frequently changing ranges, such as date-time values. These types of columns can result in old values in the HIGH2KEY and LOW2KEY columns in the

catalog. By periodically collecting column statistics on these changing columns, you can make the values in HIGH2KEY and LOW2KEY accurately reflect the range of data values. With these accurate values, DB2 can obtain accurate filter factors for range predicates.

- When you collect statistics for indexes, you can specify the FREQVAL option to specify whether distribution statistics are collected, and number of concatenated index columns to collect. When you collect statistics about indexes, cardinality statistics (including intermediate key cardinality values) are automatically collected on the specified indexes. You can also specify the FREQVAL option to specify whether distribution statistics are collected, and number of concatenated index columns to collect. By default, distribution statistics are collected on the first column of each index for the 10 most frequently occurring values. FIRSTKEYCARDF and FULLKEYCARDF are also collected by default.
- For DPSI type indexes, consider updating the FULLKECARDF value in the catalog if you know the accurate value. The FULLKEYCARDF value that is generated by DB2 utilities is an estimate that is based on a sampling method.

Related concepts:

Predicate filter factors

Related tasks:

Modifying catalog statistics to influence access path selection

Related reference:

Statistics used for access path selection

Reducing the cost of collecting statistics

You can reduce the cost of maintaining statistics for your database objects by taking certain actions.

Procedure

To reduce the cost of collecting statistics for data objects, use any of the following approaches:

- For tables that contain many rows of data, use the TABLESAMPLE AUTO option, in most cases. The TABLESAMPLE option reduces the number of rows and pages scanned. The TABLESAMPLE option can be hundreds of times faster because the number of pages and rows scanned can be 10,000 times fewer, and the number of rows sampled can be 100 times fewer. Under the default of TABLESAMPLE AUTO, DB2 determines the sampling rate based on the number of rows in a table. If real-time statistics indicate that a table contains fewer than 500,000 rows, sampling is not used to ensure the accuracy of the result.

  The TABLESAMPLE option is preferred over the SAMPLE option because the SAMPLE option does not reduce the number of rows scanned, and because the sampling rate can cause inaccurate results for tables with fewer rows.
- Consider running several RUNSTATS jobs concurrently against different partitions of a partitioned table space or index rather than running a single RUNSTATS job on the entire table space or index. The sum of the processor time for the concurrent jobs is roughly equivalent to the processor time for running the single RUNSTATS job. However, the total elapsed time for the concurrent jobs can be significantly less than when you run RUNSTATS on an entire table space or index.
- When you collect statistics at the table level, collect statistics only for columns or column groups that might be used as search conditions in WHERE clauses.
  - Specify the COLUMN option to collect statistics for specified columns only.

  - Specify the COLGROUP option to collect statistics for specified groups of columns only. You can also specify the FREQVAL option with the COLGROUP option to collect distribution statistics for the column group. You cannot specify the COLGROUP option when you collect inline statistics by using utilities other than RUNSTATS.

  When you collect statistics on groups of columns that are used in predicates, the improved accuracy of the filter factor estimate can lead to improved query performance. However, collecting statistics on all columns of a table is costly and might be unnecessary.
- Collect frequency and cardinality statistics only when the values change. For example, a column on GENDER is likely to have a COLCARDF of 2, with M and F as the possible values. It is unlikely that the cardinality for this column ever changes. The distribution of the values in the column might not change often, depending on the volatility of the data.
- Avoid running the RUNSTATS utility by specifying the STATISTICS keyword to collect inline statistics when you run any of the following utilities:
  - LOAD
  - REBUILD INDEX
  - REORG INDEX
  - REORG TABLESPACE

  When you specify STATISTICS in one of these utility statements, DB2 updates the catalog with table space or index space statistics for the objects on which the utility is run. However, you cannot collect column group statistics with the STATISTICS keyword. You can collect column group statistics only by running the RUNSTATS utility. If you restart a LOAD or REBUILD INDEX job that uses the STATISTICS keyword, DB2 does not collect inline statistics. For these cases, you must run the RUNSTATS utility after the restarted utility job completes.

Related concepts:

Restart of REORG TABLESPACE (DB2 Utilities)

Related reference:

RUNSTATS (DB2 Utilities)

LOAD (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

Automating statistics maintenance

You can reduce the amount of labor-intensive work for identifying, collecting, and maintaining accurate statistics in DB2. Doing so might also improve performance by increasing the likelihood that accurate statistics are available when DB2 needs them.

Before you begin

PSPI

- Configure the following stored procedures (use job DSNTIJRT, as described in Installing DB2-supplied routines during installation (DB2 Installation and Migration))
  - ADMIN_COMMAND_DB2
  - ADMIN_INFO_SSID
  - ADMIN_TASK_ADD
  - ADMIN_TASK_UPDATE
  - ADMIN_UTL_EXECUTE
  - ADMIN_UTL_MODIFY
  - ADMIN_UTL_MONITOR
  - ADMIN_UTL_SCHEDULE
  - ADMIN_UTL_SORT
  - DSNUTILU
  - DSNWZP
- Configure the following user-defined functions (use job DSNTIJRT):
  - ADMIN_TASK_LIST()
  - ADMIN_TASK_STATUS()
- Ensure that your authorization ID has the following privileges:
  - CALL for the preceding stored procedures and user-defined functions.
  - Read and modify data in the following catalog tables:
    - SYSIBM.SYSAUTOALERTS
    - SYSIBM.SYSAUTORUNS_HIST
    - SYSIBM.SYSAUTOTIMEWINDOWS
    - SYSIBM.SYSTABLES_PROFILES
  - Read data in the following catalog tables:
    - SYSIBM.SYSTABLESPACESTATS
    - SYSIBM.SYSTABLESPACE
    - SYSIBM.SYSDATABASE
    - SYSIBM.SYSTABLES
    - SYSIBM.SYSINDEXES
    - SYSIBM.SYSKEYS
    - SYSIBM.SYSCOLUMNS
    - SYSIBM.SYSCOLDIST
    - SYSIBM.SYSDUMMY1
    - SYSIBM.UTILITY_OBJECTS

About this task

Whether you configure autonomic monitoring directly within DB2, or use an administrative tool outside of DB2, the high-level steps for configuring autonomic monitoring are similar.

After you configure autonomic monitoring, DB2 relies on scheduled calls to the ADMIN_UTL_MONITOR stored procedure to monitor your statistics. When stale, missing, or conflicting, are identified, the ADMIN_UTL_EXECUTE stored procedure invokes RUNSTATS within defined maintenance windows and resolves the problems. The ADMIN_UTL_EXECUTE stored procedure uses the options that are defined in *statistics profiles* to invoke the RUNSTATS utility. The

ADMIN_UTL_MODIFY stored procedure is called at regular intervals to clean up the log file and alert history.

Procedure

To configure autonomic monitoring:

1. Schedule time windows for autonomic statistics collection. The time windows are defined in the SYSIBM.SYSAUTOTIMEWINDOWS catalog table.
2. Schedule monitoring activities and define the level of detail, thresholds, objects to exclude from monitoring, and other options. The monitoring activities are scheduled through one or more tasks defined in the administrative task scheduler. Each task calls the ADMIN_UTL_MONITOR stored procedure and can define a different level of detail, different thresholds, and different objects to exclude from monitoring.
3. Schedule maintenance of the log and alert history for autonomic statistics. The monitoring activities are scheduled through a single task that is defined in the administrative task scheduler that calls the ADMIN_UTL_MODIFY stored procedure.

PSPI

Related concepts:

Statistics profiles

Related tasks:

Combining autonomic and manual statistics maintenance

Related reference:

ADMIN_UTL_EXECUTE stored procedure (DB2 SQL)

ADMIN_UTL_MODIFY stored procedure (DB2 SQL)

ADMIN_UTL_MONITOR stored procedure (DB2 SQL)

Autonomic statistics overview

DB2 uses interactions between the administrative scheduler, certain DB2-supplied stored procedures, and certain catalog tables for autonomic statistics maintenance.

PSPI

The following diagram illustrates the relationships between the various objects that DB2 uses for autonomic statistics maintenance. DB2 uses the following actions for autonomic statistics maintenance:

1. The administrative task scheduler issues calls to the ADMIN_UTL_MONITOR stored procedure according to the schedule that you specify.
2. When the ADMIN_UTL_MONITOR detects missing, out-of-date, or conflicting statistics, it issues a call to the ADMIN_TASK_ADD stored procedure to schedule an immediate execution of the ADMIN_UTL_EXECUTE stored procedure.
3. The administrative scheduler calls the ADMIN_UTL_EXECUTE stored procedure.
4. When the call to ADMIN_UTL_EXECUTE stored procedure occurs within a time window that you specify, it invokes the RUNSTATS utility to solve alerts.

5. When the call to the ADMIN_UTL_EXECUTE stored procedure occurs outside of a specified time window, the ADMIN_UTL_EXECUTE stored procedure issues a call to the ADMIN_TASK_ADD stored procedure to reschedule its own execution to the next time window.

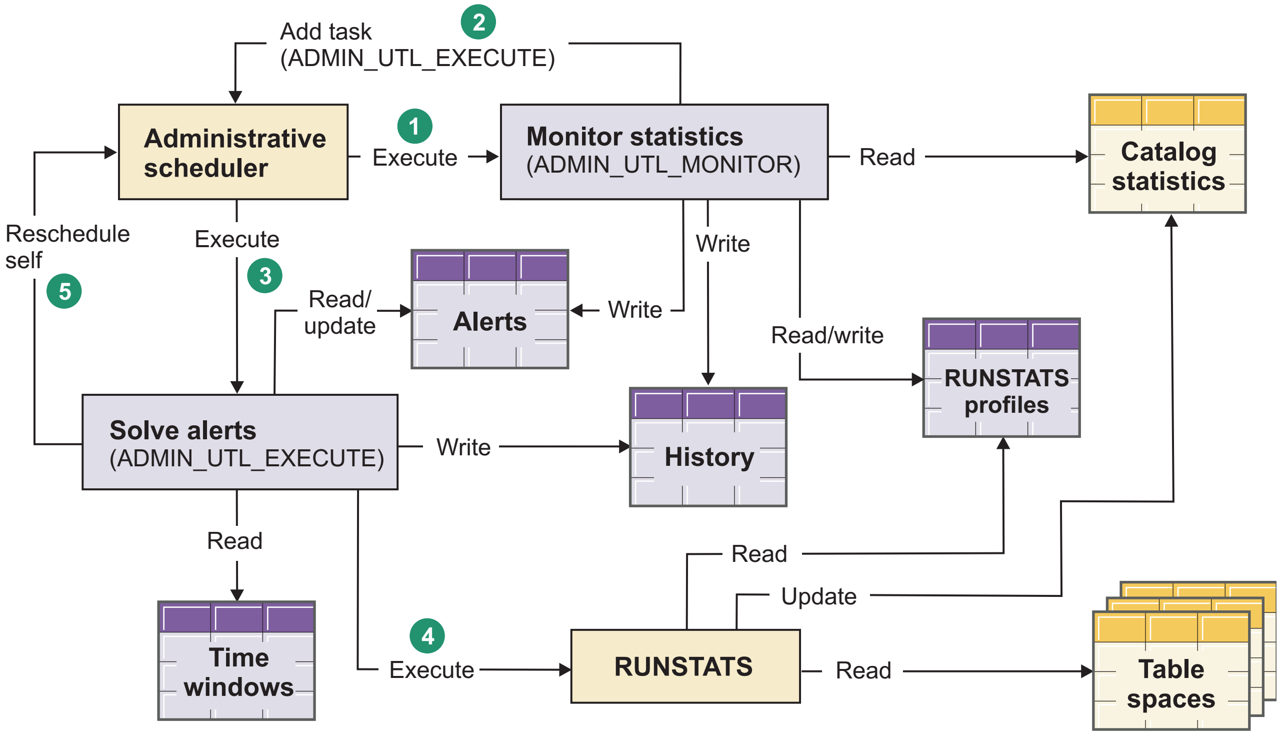


Figure 34. Object interactions for autonomic statistics maintenance in DB2

The administrative scheduler and the following stored procedures and tables have the following roles when you enable autonomic statistics in DB2:

Administrative scheduler
Calls each of the stored procedures:

- Calls ADMIN_UTL_MONITOR stored procedure as scheduled for statistics monitoring.
- Calls the ADMIN_UTL_EXECUTE stored procedure to solve alerts according to tasks added by the ADMIN_UTL_MONITOR and ADMIN_UTL_EXECUTE stored procedures.
- Calls the ADMIN_UTL MODIFY stored procedure as scheduled to remove old alert history information from the SYSIBM.SYSAUTOALERTS catalog table.

ADMIN_UTL_MONITOR stored procedure
Monitors statistics in the catalog and identifies stale, missing, or conflicting statistics through the following interactions:

- Reading catalog tables to assess existing statistics, and generate alerts as necessary
- Reading the RUNSTATS profiles to determine the set of columns, column groups, and indexes to check.
- When alerts are found, adding tasks for the ADMIN_UTL_EXECUTE stored procedure to the administrative scheduler.
- Writing statistics alerts.

ADMIN_UTL_EXECUTE stored procedure

Invokes the RUNSTATS utility to fix stale, missing or conflicting statistics that are identified by the ADMIN_UTL_MONITOR stored procedure through the following interactions:

- Reading alerts and updating alerts when they are solved.
- Reading the time windows to determine when to solve alerts.
- Adding tasks for its own execution at the next time window
- During time windows, invoking the RUNSTATS utility to collect statistics.

SYSIBM.SYSTABLES_PROFILES catalog table

Contains RUNSTATS profiles that are used to specify the options for the invocation of the RUNSTATS utility by the ADMIN_UTL_EXECUTE stored procedure. Each row creates a single profile for a different table.

SYSIBM.SYSAUTOTIMEWINDOWS catalog table

Contains information that describes time windows during which the ADMIN_UTL_EXECUTE stored procedure is allowed to invoke the RUNSTATS utility to resolve alerts.

SYSIBM.SYSAUTOALERTS catalog table

Contains information about statistics alerts that are identified by the ADMIN_UTL_MONITOR stored procedure.

SYSIBM.SYSAUTORUNS_HIST catalog table

Contains historical information about the actions of the ADMIN_UTL_MONITOR, ADMIN_UTL_EXECUTE, and ADMIN_UTL_MODIFY stored procedures.

ADMIN_UTL_MODIFY stored procedure

Reads the SYSIBM.SYSAUTORUNS_HIST and SYSIBM.SYSAUTOALERTS catalog table and deletes old records in that table based on the options that are specified when the call is scheduled in the administrative task scheduler.

PSPI

Specifying time windows for collecting autonomic statistics

You can specify time windows when the autonomic collection of statistics is allowed.

Before you begin

PSPI

Your authorization ID must have privileges to modify data in the SYSIBM.SYSAUTOTIMEWINDOWS catalog table.

About this task

When the administrative task scheduler calls the ADMIN_UTL_EXECUTE stored procedure, the ADMIN_UTL_EXECUTE stored procedure checks whether the call is within a specified time window before resolving statistics alerts. If the call is not within a time window, the ADMIN_UTL_EXECUTE stored procedure reschedules itself in the administrative task scheduler to the next specified time window.

Procedure

To specify time windows during which the ADMIN_UTL_EXECUTE stored procedure can invoke the RUNSTATS utility to resolve statistics alerts:

Insert rows into the SYSIBM.SYSAUTOTIMEWINDOWS catalog table. Each row describes a single time window for a particular DB2 member. Any row that does not specify a member (contains a NULL value) applies to all DB2 members. For example, you might issue the following INSERT statements to create the specified time windows.

Every Monday from 12 midnight to just before midnight Tuesday, allowing 10 parallel tasks

You might issue the following statement:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH,DAY, FROM_TIME, TO_TIME, ACTION, MAX_TASKS)
  VALUES(NULL,'W',NULL,1,'00:00','23:59:59',NULL,10);
```

The following statement also achieves the same result:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH, DAY, FROM_TIME, TO_TIME, ACTION, MAX_TASKS)
  values(NULL,'W',NULL,1,NULL,NULL,NULL,10);
```

Every Monday in January from 12 midnight to 12 noon, allowing 5 parallel tasks

Issue the following statement:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH, DAY, FROM_TIME, TO_TIME ,ACTION, MAX_TASKS)
  VALUES(NULL,'W',1,1,'00:00','12:00:00',NULL,5);
```

The first day of each month from 12 midnight to 12 noon, allowing 5 parallel tasks

You might issue the following statement:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH, DAY, FROM_TIME, TO_TIME ,ACTION, MAX_TASKS)
  VALUES(NULL,'M',NULL,1,'00:00','12:00:00',NULL,5);
```

Each first day of June from 08 a.m. to 2 p.m., allowing 5 parallel tasks

Issue the following statement:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH, DAY, FROM_TIME, TO_TIME ,ACTION, MAX_TASKS)
  VALUES(NULL,'M',6,1,'08:00','14:00:00',NULL,5);
```

Every Sunday from 12 midnight to 12 noon, allowing 5 parallel tasks, on member DSN1

Issue the following statement:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH, DAY, FROM_TIME, TO_TIME ,ACTION, MAX_TASKS)
  VALUES('DSN1','W',NULL,7,'00:00','12:00:00',NULL,5);
```

Every Sunday from 12 midnight to 12 noon, allowing 5 parallel tasks, on member DSN1, allowing only RUNSTATS alerts

Issue the following statement:

```
INSERT INTO SYSIBM.SYSAUTOTIMEWINDOWS(DB2_SSID, MONTH_WEEK,
 MONTH, DAY, FROM_TIME, TO_TIME ,ACTION, MAX_TASKS)
  VALUES('DSN1','W',NULL,7,'00:00','12:00:00','RUNSTATS',5);
```

PSPI

Related reference:

ADMIN_UTL_EXECUTE stored procedure (DB2 SQL)

SYSIBM.SYSAUTOTIMEWINDOWS table (DB2 SQL)

Scheduling autonomic statistics monitoring

You can schedule maintenance activities that run automatically to maintain accurate and up-to-date statistics in DB2.

Before you begin

Your authorization ID must have execute privileges for the ADMIN_TASK_ADD stored procedure.

The authorization ID that you specify in the call to the ADMIN_TASK_ADD stored procedure must have ADMIN_UTL_MONITOR stored procedure.

Procedure

PSPI

To schedule autonomic statistics monitoring:

Call the ADMIN_TASK_ADD stored procedure to schedule calls to the ADMIN_UTL_MONITOR stored procedures in the administrative task scheduler. The following examples show the relevant values that you might specify in the call to the ADMIN_TASK_ADD stored procedure:

Monitor statistics, excluding the catalog database, every day at 1 a.m.

The following option values define the described schedule:

point-in-time

```
pstmt.setString(7, "0 1 * * *");
```

procedure-name

```
pstmt.setString(13, "ADMIN_UTL_MONITOR");
```

procedure-input

```
pstmt.setString(14, "SELECT 'statistics-scope=profile,restrict-
ts=\"DBNAME <> ''DSNDB06''\"', 0, 0 ,'' FROM SYSIBM.SYSDUMMY1");
```

Monitor statistics for the SYSTSKEY catalog table space on the first day of each month at 1 a.m.

The following option values define the described schedule:

point-in-time

```
pstmt.setString(7, "0 1 1 * *");
```

procedure-name

```
pstmt.setString(13, "ADMIN_UTL_MONITOR");
```

procedure-input

```
pstmt.setString(14, "SELECT 'statistics-scope=profile,
restrict-ts=\"DBNAME = ''DSNDB06'' AND NAME =''SYSTSKEY''\"', 0,
0 ,'' FROM SYSIBM.SYSDUMMY1");
```

PSPI

Related tasks:

Scheduling administrative tasks (DB2 Administration Guide)

Related reference:

ADMIN_TASK_ADD stored procedure (DB2 SQL)

ADMIN_UTL_MONITOR stored procedure (DB2 SQL)

Defining the scope of autonomic statistics monitoring

You can specify tables to include and exclude from autonomic statistics monitoring.

About this task

PSPI

You might want to exclude tables from autonomic monitoring for certain special situations, such as those that require you to maintain or manipulate statistics manually, or for particularly large tables.

Procedure

To define the scope of autonomic monitoring:

Specify the restrict-ts option when you schedule calls to the ADMIN_UTL_MONITOR stored procedure in the administrative scheduler. The restrict-ts option allows any string that contains valid content for a WHERE clause on the SYSIBM.SYSTABLESPACESTATS catalog table. For example, you might specify the following predicates for the restrict-ts option:

- To check only the DB2 catalog:
  ```
  DBNAME='DSNDB06'
  ```
- To check all table spaces, except for the DB2 catalog:
  ```
  DBNAME<>'DSNDB06'
  ```
- To check all tables spaces that have names that begin with 'A':
  ```
  NAME LIKE 'A%'
  ```
- To check all table spaces that were created by SYSIBM:
  ```
  (DBNAME,NAME) in (SELECT DBNAME,TSNAME
  FROM SYSIBM.SYSTABLES WHERE
  CREATOR='SYSIBM')
  ```

Only tables spaces that meet the criteria of the predicates that are defined in the restrict-ts option are checked by autonomic statistics. When the restrict-ts option is not specified, all table spaces are checked by autonomic statistics.

PSPI

Related reference:

ADMIN_UTL_MONITOR stored procedure (DB2 SQL)

Scheduling log and alert history cleanup for autonomic statistics

When you automate statistics maintenance, DB2 stores logs and historical alert information. You can schedule autonomic cleanup and removal of these records.

Before you begin

PSPI

The following prerequisites are met:

- For scheduling the cleanup, your authorization ID has execute privileges for the ADMIN_TASK_ADD stored procedure, and the authorization ID that you specify in the call to the ADMIN_TASK_ADD stored procedure has execute privileges for the ADMIN_UTL_MODIFY stored procedure.
- When invoking the cleanup manually, your authorization ID has execute privileges on the ADMIN_UTL_MODIFY stored procedure.

About this task

DB2 logs information about autonomic statistics activities in the SYSIBM.SYSAUTOALERTS and SYSIBM.SYSAUTORUNST_HIST catalog tables. The activity logs include summary and detailed reports about generated alerts and the associated executions of the autonomic stored procedures.

Procedure

To schedule cleanup activities for the logs and alert histories for autonomic statistics:

Add a single task that executes the ADMIN_UTL_MODIFY stored procedure to the administrative task scheduler. The following example shows the relevant values that you might specify in the call to the ADMIN_TASK_ADD stored procedure to schedule the removal of log and alert history data that is older than thirty days, on the first day of every month:

point-in-time

```
pstmt.setString(7, "0 0 1 * * ");
```

procedure-name

```
pstmt.setString(13, "ADMIN_UTL_MODIFY");
```

procedure-input

```
pstmt.setString(14, SELECT 'history-days=30', 0, 0, '' FROM
SYSIBM,SYSDUMMY1");
```

You can specify the frequency of the execution and other options by defining the task properties.

PSPI

Related tasks:

Scheduling administrative tasks (DB2 Administration Guide)

Related reference:

ADMIN_UTL_MODIFY stored procedure (DB2 SQL)

ADMIN_TASK_ADD stored procedure (DB2 SQL)

Statistics profiles

A *statistics profile* is a saved set of options for the RUNSTATS utility. Each profile applies for a particular table. DB2 uses statistics profiles for autonomic statistics maintenance. You can also use statistics profiles to quickly run the RUNSTATS utility with a predefined set of options.

You can specify a complete set of RUNSTATS options in a profile, or specify only a few options, or even only a single option. The options that you specify are stored in the PROFILE_TEXT column of the SYSIBM.SYSTABLES_PROFILES catalog table. If you do not specify values for the following options when you create the profile, DB2 uses default values, as in any RUNSTATS invocation:

- COLUMN
- COLGROUP
- FREQVAL
- COUNT
- MOST
- BOTH
- LEAST
- INDEX
- KEYCARD
- NUMCOLS
- COUNT
- HISTOGRAM
- NUMQUANTILES

Each statistics profile is saved as a single row in the SYSIBM.SYSTABLES_PROFILES catalog table. After your create a profile for a table, you can specify that DB2 uses the same options that were specified in the table when you collect statistics again later. When you automate statistics maintenance, DB2 creates or updates the single profile for each table that is not excluded from autonomic maintenance. Because only a single statistics profile can exist for each table, DB2 uses any options that you have specified in existing profiles for a particular table when the ADMIN_UTL_MONITOR stored procedure first executes for autonomic statistics monitoring.

Regardless of whether profiles exist, or whether autonomic statistics maintenance is enabled, you can always use utilities to collect statistics and specify customized options without using a profile.

Related tasks:

Automating statistics maintenance

Related reference:

Statistics profile syntax (DB2 Utilities)

SYSIBM.SYSTABLES_PROFILES table (DB2 SQL)

RUNSTATS (DB2 Utilities)

Creating statistics profiles

You can create and use *statistics profiles* to collect statistics for particular tables with consistent options, without the need to explicitly specify the options each time. DB2 also uses the statistics profiles when you implement autonomic statistics maintenance.

About this task

DB2 uses statistics profiles when you enable autonomic statistics maintenance. When you first enable autonomic statistics maintenance, the ADMIN_UTL_MONITOR stored procedure sets a profile for each monitored table based on the existing statistics. However, if a profile exists for a table, DB2 uses the existing profile.

Procedure

To set a statistics profile, complete one of the following actions:

- Issue the following utility control statement to explicitly specify collection options for the profile:

  ```
  RUNSTATS TABLESPACE ts-name TABLE table-name runstats-options SET PROFILE
  ```

  DB2 records the values specified by *runstats-options* in the PROFILE_TEXT column of the SYSIBM.SYSTABLES_PROFILES catalog table. DB2 uses the default values for any options that are not specified.
- Issue the following utility control statement to automatically specify options in the profile based on the existing statistics for the specified table:

  ```
  RUNSTATS TABLESPACE ts-name TABLE table-name SET PROFILE FROM EXISTING STATS
  ```

  The generated profile is based on the statistics that exist in the catalog tables. However, the keywords used in the generated profile do not necessarily match the keywords that were used to collect the statistics previously.

Results

No statistics are collected when you run the RUNSTATS utility with the SET PROFILE option. If a profile exists for the specified table, that profile is replaced with the new one, and the existing profile is lost. Only one profile can exist for a particular table.

Related reference:

Statistics profile syntax (DB2 Utilities)

RUNSTATS (DB2 Utilities)

Related information:

Collecting statistics by using statistics profiles

You can use statistics profiles to invoke the RUNSTATS utility with a predefined set of statistics collection options. DB2 also uses statistics profiles when you enable autonomic statistics maintenance.

Procedure

To collect statistics by using a statistics profile:

Issue the following RUNSTATS control statement:

```
RUNSTATS TABLESPACE ts-name TABLE table-name USE PROFILE
```

DB2 collects statistics for the table specified by *table-name* according to the collection options that are specified in the profile, and issues a message to indicate that the profile was used. If no profile exists for a target table, DB2 issues an error message.

Related tasks:

Automating statistics maintenance

Creating statistics profiles

Related information:

DSNU1364I (DB2 Messages)

Updating statistics profiles

You can modify options to change the statistics that are collected by existing statistics profiles that you have created, or those that are created for autonomic statistics monitoring by the ADMIN_UTL_MONITOR stored procedure.

Procedure

To modify an existing statistics profile:

Issue following RUNSTATS control statement:

```
RUNSTATS TABLESPACE ts-name TABLE table-name runstats-options UPDATE PROFILE
```

Results

No statistics are collected when you invoke the RUNSTATS utility with the UDPATE PROFILE option. DB2 replaces any existing options that are specified in PROFILE_TEXT column of the SYSIBM.SYSTABLES_PROFILES catalog table with values options that are specified in *runstats-options* for the table that is specified by *table-name*. Any options that are not specified remain unchanged in the existing profile. If no profile exists for the specified table, DB2 issues an error message.

Related reference:

Statistics profile syntax (DB2 Utilities)

Related information:

DSNU1363I (DB2 Messages)

Deleting statistics profiles

You can delete an existing statistics profile.

Procedure

To delete existing RUNSTATS profiles:

Issue the following utility control statement:

```
RUNSTATS TABLESPACE ts-name TABLE options DELETE PROFILE
```

Any existing profile for the table or tables specified in *options* is removed. No statistics are collected when you invoke the RUNSTATS utility with the DELETE PROFILE option. If no profile exists for the specified table or tables DB2 issues an error message.

Related reference:

Statistics profile syntax (DB2 Utilities)

Related information:

DSNU1363I (DB2 Messages)

Combining autonomic and manual statistics maintenance

When autonomic statistics maintenance is enabled, you can still invoke the RUNSTATS utility to capture statistics manually.

About this task

When autonomic statistics monitoring and maintenance is enabled, DB2 uses statistics profiles to maintain the statistics for each table that is not excluded from autonomic maintenance. However, you can still explicitly invoke the RUNSTATS utility at any time either in the traditional manner, or by using profiles at any time.

Procedure

To effectively combine autonomic and manual statics maintenance activities, you might follow the following recommendations:

- Before enabling autonomic statistics maintenance, consider whether to delete all existing statistics profiles by issuing RUNSTATS control statements and specifying the DELETE PROFILE option. By doing that, you enable DB2 to create new statistics profiles based on analysis of your existing statistics. However, this step is optional if you prefer that DB2 uses the settings of your existing RUNSTATS profiles for autonomic maintenance.
- When you want to collect statistics with different settings than those that are used for autonomic maintenance, use the traditional method for invoking the RUNSTATS utility and explicitly specify the options that you want to use. Invoking RUNSTATS in that manner has no impact on the options that are specified in the profile, and periodic autonomic maintenance can continue unchanged. However, because the manual invocation of RUNSTATS does not change the RUNSTATS profile, the manually collected might be lost at the next invocation of RUNSTATS that uses the profile. Consequently, you might want to update the profile to use the new options that were specified in the manual invocation.
- When you want to manually invoke the collection of statistics outside of the autonomic maintenance windows, but with the usual settings, you can specify the USE PROFILE option in the RUNSTATS control statement.
- When you want to modify the settings that are used for autonomic maintenance, you can issue a RUNSTATS control statement with the UPDATE PROFILE option and specify that options that you want to change. After you update the profile, DB2 uses the new options for autonomic maintenance activities.

Related tasks:

Automating statistics maintenance

Deleting statistics profiles

Related reference:

- RUNSTATS (DB2 Utilities)
- RUNSTATS TABLESPACE syntax and options (DB2 Utilities)
- Statistics profile syntax (DB2 Utilities)

Collecting histogram statistics

You can enable improved access path selection by collecting histogram statistics.

About this task

DB2 uses histogram statistics to estimate predicate selectivity from value-distribution statistics that are collected over the entire range of values in a data set.

The recommendation is to collect histogram statistics for columns that are specified in predicates, ORDER BY, GROUP BY, and HAVING clauses of SQL statements. Histogram statistics are most helpful for range, LIKE, and BETWEEN predicates, but they can also improve selectivity for equality, IS NULL, and IN predicates, and predicates that compare two columns.

The collection of histogram requires a sort operation. In the event that FREQVAL statistics are also collected, the same sort operation is used. For indexes with columns of mixed order, statistics can be collected only for the prefix columns of the same order.

Procedure

To collect histogram statistics:

Optional: Specify an integer value for the NUMQUANTILES option to set a guideline for number of intervals that DB2 uses for the histogram statistics. The value must be greater than or equal to one, and less than or equal to 100.
The recommendation is to not specify the value of NUMQUANTILES in most cases. When the NUMQUANTILES value is not specified, DB2 uses as many as 100 intervals. However, DB2 might collect a smaller number of intervals, which is optimized for the number of rows and the number of distinct values in the table, and other factors.
You might want to specify a specific value for NUMQUANTILES for certain applications situations that are well understood. For example, if queries frequently specify range intervals such as 0-10%, 10-20%, 20-30%, and so on, then a NUMQUANTILES value of 10 might be more appropriate.
Even when you specify a value for NUMQUANTILES, the exact number of resulting intervals that DB2 creates is likely to vary from the value that you specify, depending on the number of rows, the number of distinct values, and other factors.

Related concepts:

Histogram statistics

Histogram statistics filter factors

Related reference:

- RUNSTATS (DB2 Utilities)
- SYSIBM.SYSCOLDIST table (DB2 SQL)
- SYSIBM.SYSKEYTGTDIST table (DB2 SQL)

Related information:

Histogram statistics over a range of column values (DB2 9 for z/OS Performance Topics)

Histogram statistics (DB2 9 for z/OS Performance Topics)

Histogram statistics recommendations (DB2 9 for z/OS Performance Topics)

Histogram statistics

Histogram statistics enable DB2 to improve access path selection by estimating predicate selectivity from value-distribution statistics that are collected over the entire range of values in a data set.

Restriction: RUNSTATS cannot collect histogram statistics on randomized key columns.

DB2 chooses the best access path for a query based on predicate selectivity estimation, which in turn relies heavily on data distribution statistics. Histogram statistics summarize data distribution on an interval scale by dividing the entire range of possible values within a data set into a number of intervals.

The following values define the intervals:

QUANTILENO
An ordinary sequence number that identifies the interval.

HIGHVALUE
The value that serves as the upper bound for the interval.

LOWVALUE
A value that serves as the lower bound for the interval.

DB2 creates histogram statistics in the following manner:

- The first interval contains the lowest value for a column or column group.
- The last interval contains the highest value for a column or column group.
- If the column is nullable and contains NULL values, the NULL value and second-highest value are each placed in separate intervals.
- The remaining intervals contain approximately equal-depth histogram statistics, which means that the whole range of values is divided into intervals that each contain about the same percentage of the total number rows.

Histogram statistics intervals

Histogram statistics intervals have the following characteristics:

- A highly frequent single value might occupy an interval by itself.
- A single value is never broken into more than one interval, meaning that the maximum number of intervals is equal to the number of distinct values on the column. The maximum number of intervals cannot exceed 100, which is the maximum number that DB2 supports.
- Adjacent intervals sometime skip values that do not appear in the table, especially when doing so avoids a large range of skipped values within an interval. For example, if the value 30 above has 1% frequency, placing it in the seventh interval would balance the percentage of rows in the 6th and 7th intervals. However, doing so would introduce a large skipped range to the seventh interval.

- HIGHVALUE and LOWVALUE can be inclusive or exclusive, but an interval generally represents a non-overlapped value range.
- NULL values, if any exist, occupy a single interval.
- Because DB2 cannot break any single value into two different intervals, the maximum number of intervals is limited to the number of distinct values in the column, and cannot exceed the supported maximum of 100 intervals.

Related concepts:

Histogram statistics filter factors

Related tasks:

Collecting histogram statistics

Related reference:

RUNSTATS (DB2 Utilities)

SYSIBM.SYSCOLDIST table (DB2 SQL)

SYSIBM.SYSKEYTGTDIST table (DB2 SQL)

Related information:

Histogram statistics over a range of column values (DB2 9 for z/OS Performance Topics)

Histogram statistics (DB2 9 for z/OS Performance Topics)

Histogram statistics recommendations (DB2 9 for z/OS Performance Topics)

Collecting statistics by partition

You can collect statistics for a single data partition or index partition to avoid the cost of running utilities against unchanged partitions.

About this task

For a partitioned table space, DB2 keeps statistics separately by partition and also collectively for the entire table space. The following table shows the catalog tables that contain statistics by partition and, for each one, the table that contains the corresponding aggregate statistics.

Table 76. The catalog tables that contain statistics by partition and the table that contains the corresponding aggregate statistics

Statistics by partition are in	Aggregate statistics are in
SYSTABSTATS	SYSTABLES
SYSINDEXSTATS	SYSINDEXES
SYSCOLSTATS	SYSCOLUMNS
SYSCOLDISTSTATS	SYSCOLDIST
SYSKEYTARGETSTATS	SYSKEYTARGETS
SYSKEYTGTDISTSTATS	SYSKEYTGTDIST

Procedure

To collect statistics by partition:

Specify the PART option when you collect statistics with the RUNSTATS utility or inline with another utility.

When you run utilities by partition, DB2 uses the results to update the aggregate statistics for the entire table space or index. If statistics do not exist for each separate partition, DB2 can calculate the aggregate statistics only if the utilities are run under one of the following options:

- The FORCEROLLUP YES option is specified when the utility is run.
- The value of the STATROLL subsystem parameter is YES.

If you do not use specify these options, you must collect statistics on the entire object before you collect statistics on separate partitions.

Related tasks:

Collecting statistics by using DB2 utilities

Related reference:

STATISTICS ROLLUP field (STATROLL subsystem parameter) (DB2 Installation and Migration)

SYSIBM.SYSKEYTARGETS table (DB2 SQL)

SYSIBM.SYSKEYTARGETSTATS table (DB2 SQL)

SYSIBM.SYSTABSTATS table (DB2 SQL)

Collecting history statistics

You can save statistics history information in catalog history tables.

Procedure

To save history statistics in the catalog:

1. Specify the HISTORY option when you run the RUNSTATS utility or collect inline statistics with another utility. When you specify the HISTORY option, the utility stores the statistics that were updated in the catalog tables in history records in the corresponding catalog history tables.
2. Remove old statistics from the catalog history tables by taking one of the following actions:
   - Run the MODIFY STATISTICS utility.
   - Issue a DELETE statement.

   Deleting outdated information from the catalog history tables can help improve the performance of processes that access the data in these tables.

Related concepts:

Guidelines for deciding which statistics history rows to delete (DB2 Utilities)

Deletion of specific statistics history rows (DB2 Utilities)

Related tasks:

Collecting statistics history (DB2 Utilities)

Related reference:

STATISTICS HISTORY field (STATHIST subsystem parameter) (DB2 Installation and Migration)

RUNSTATS (DB2 Utilities)

LOAD (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

MODIFY STATISTICS (DB2 Utilities)

History statistics

Certain catalog tables contain historical statistics about activity in other catalog tables.

PSPI

When DB2 adds or changes rows in a catalog table, DB2 might also write information about the rows to a corresponding *catalog history table*. Although the catalog history tables are not identical to their counterpart tables, they do contain the same columns for access path information and space utilization information. The history statistics provide a way to study trends, to determine when utilities, such as REORG, should be run for maintenance, and to aid in space management.

Each catalog history table bears the name of the catalog table that it describes appended by "_HIST". The following tables describe some of the relevant columns in the catalog history tables:

SYSIBM.SYSCOLDIST_HIST

Table 77. Historical statistics columns in the SYSCOLDIST_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CARDF	Yes	No	For TYPE C, Number of distinct values gathered in the column group; for TYPE='H', the number of distinct values in the column group of the interval indicated by the value in the QUANTILENO column
COLGROUPCOLNO	Yes	No	Identifies the columns involved in multi-column statistics
COLVALUE	Yes	No	Frequently occurring value in the key distribution
FREQUENCYF	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	No	For TYPE='H', the value of the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	No	For TYPE='H', the value of the low bound for the interval indicated by the value of the QUANTILENO column.
NUMCOLUMNS	Yes	No	Number of columns involved in multi-column statistics

Table 77. Historical statistics columns in the SYSCOLDIST_HIST catalog table (continued)

Column name	Provides access path statistics[1]	Provides space statistics	Description
TYPE	Yes	No	The type of statistics gathered: **C** Cardinality **F** Frequent value **P** Non-padded **H** Histogram statistics
QUANTILENO	Yes	No	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.

SYSIBM.SYSCOLUMNS_HIST

Table 78. Historical statistics columns in the SYSCOLUMNS_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
COLCARDF	Yes	No	Estimated number of distinct values in the column
HIGH2KEY	Yes	No	Second highest value of the column, or blank
LOW2KEY	Yes	No	Second lowest value of the column, or blank

SYSIBM.SYSINDEXES_HIST

Table 79. Historical statistics columns in the SYSINDEXES_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CLUSTERING	Yes	No	Whether the index was created with CLUSTER
CLUSTERRATIOF	Yes	No	A number, when multiplied by 100, gives the percentage of rows in the clustering order.
FIRSTKEYCARDF	Yes	No	Number of distinct values in the first key column
FULLKEYCARDF	Yes	No	Number of distinct values in the full key
NLEAF	Yes	No	Number of active leaf pages
NLEVELS	Yes	No	Number of levels in the index tree
DATAREPEATFACTORF	Yes	No	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.

SYSIBM.SYSINDEXPART_HIST

Table 80. Historical statistics columns in the SYSINDEXPART_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CARDF	No	No	Number of rows or LOBs referenced.

Table 80. Historical statistics columns in the SYSINDEXPART_HIST catalog table (continued)

Column name	Provides access path statistics[1]	Provides space statistics	Description
DSNUM	No	Yes	Number of data sets
EXTENTS	No	Yes	Number of data set extents (for multiple pieces, the value is for the extents in the last data set)
FAROFFPOSF	No	Yes	Number of rows referenced far from the optimal position.
LEAFDIST	No	Yes	100 times the number of pages between successive leaf pages
LEAFFAR	No	Yes	Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan
LEAFNEAR	No	Yes	Number of leaf pages located physically near previous leaf pages for successive active leaf pages
NEAROFFPOSF	No	Yes	Number of rows referenced near but not at the optimal position.
PQTY	No	Yes	Primary space allocation in 4K blocks for the data set
PSEUDO_DEL_ENTRIES	No	Yes	Number of pseudo-deleted keys
SECQTYI	No	Yes	Secondary space allocation in 4K blocks for the data set.
SPACEF	No	Yes	Disk storage in KB

SYSIBM.SYSINDEXSTATS_HIST

Table 81. Historical statistics columns in the SYSINDEXSTATS_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CLUSTERRATIOF	Yes	No	A number, which when multiplied by 100, gives the percentage of rows in the clustering order.
FIRSTKEYCARDF	Yes	No	Number of distinct values of the first key column
FULLKEYCARDF	Yes	No	Number of distinct values of the full key
KEYCOUNTF	Yes	No	Total number of rows in the partition.
NLEAF	Yes	No	Number of leaf pages
NLEVELS	Yes	No	Number of levels in the index tree
DATAREPEATFACTORF	Yes	No	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.

SYSIBM.SYSKEYTARGETS_HIST

Table 82. Historical statistics columns in the SYSKEYTARGETS_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
KEYCARDF	Yes	No	For type C statistics, the number of distinct values for key-target
HIGH2KEY	Yes	No	The second highest key value
LOW2KEY	Yes	No	The second lowest key value
STATS_FORMAT	Yes	No	Type of statistics gathered: **blank** No statistics have been collected, or VARCHAR column statistical values are padded **N** Varchar statistical values are not padded

SYSIBM.SYSKEYTGTDIST_HIST

Table 83. Historical statistics columns in the SYSKEYTGTDIST_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CARDF	Yes	No	For TYPE C, Number of distinct values gathered in the key group; for TYPE='H', the number of distinct values in the key group of the interval indicated by the value in the QUANTILENO column
KEYGROUPKEYNO	Yes	No	Identifies the keys involved in multi-column statistics
KEYVALUE	Yes	No	Frequently occurring value in the key distribution
HIGHVALUE	Yes	No	For TYPE='H', the value of the high bound for the interval indicated by the value of the QUANTILENO column.
FREQUENCYF	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of KEYVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	No	For TYPE='H', the value of the low bound for the interval indicated by the value of the QUANTILENO column.
NUMKEYS	Yes	No	Number of keys involved in multi-key statistics
TYPE	Yes	No	The type of statistics gathered: **C** Cardinality **F** Frequent value **P** Non-padded **H** Histogram statistics
QUANTILENO	Yes	No	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.

SYSIBM.SYSLOBSTATS_HIST

Table 84. Historical statistics columns in the SYSLOBSTATS_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
FREESPACE	No	Yes	The amount of free space in the LOB table space
ORGRATIO	No	Yes	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized. A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space has an ORGRATIO value of 100.00.

SYSIBM.SYSTABLEPART_HIST

Table 85. Historical statistics columns in the SYSTABLEPART_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CARDF	No	Yes	Number of rows in the table space or partition
DSNUM	No	Yes	Number of data sets
EXTENTS	No	Yes	Number of data set extents (for multiple pieces, the value is for the extents in the last data set)
FARINDREF	No	Yes	Number of rows relocated far from their original position
NEARINDREF	No	Yes	Number of rows relocated near their original position
PAGESAVE	No	Yes	Percentage of pages saved by data compression
PERCACTIVE	No	Yes	Percentage of space occupied by active pages
PERCDROP	No	Yes	Percentage of space occupied by pages from dropped tables
PQTY	No	Yes	Primary space allocation in 4K blocks for the data set
SECQTYI	No	Yes	Secondary space allocation in 4K blocks for the data set.
SPACEF	No	Yes	The number of KB of space currently used

SYSIBM.SYSTABLES_HIST

Table 86. Historical statistics columns in the SYSTABLES_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
AVGROWLEN	No	Yes	Average row length of the table specified in the table space
CARDF	Yes	No	Number of rows in the table or number of LOBs in an auxiliary table
NPAGESF	Yes	No	Number of pages used by the table

Table 86. Historical statistics columns in the SYSTABLES_HIST catalog table (continued)

Column name	Provides access path statistics[1]	Provides space statistics	Description
PCTPAGES	No	Yes	Percentage of pages that contain rows
PCTROWCOMP	Yes	No	Percentage of active rows compressed

SYSIBM.SYSTABSTATS_HIST

Table 87. Historical statistics columns in the SYSTABSTATS_HIST catalog table

Column name	Provides access path statistics[1]	Provides space statistics	Description
CARDF	Yes	No	Number of rows in the partition
NPAGES	Yes	No	Total number of pages with rows

Notes:

1. The access path statistics in the history tables are collected for historical purposes and are not used for access path selection.

PSPI

Related concepts:

Guidelines for deciding which statistics history rows to delete (DB2 Utilities)

Deletion of specific statistics history rows (DB2 Utilities)

Related tasks:

Collecting statistics history (DB2 Utilities)

Related reference:

STATISTICS HISTORY field (STATHIST subsystem parameter) (DB2 Installation and Migration)

MODIFY STATISTICS (DB2 Utilities)

RUNSTATS (DB2 Utilities)

Setting default statistics for created temporary tables

If you can estimate the normal cardinality and number of pages that a particular created temporary table uses, you can set the default values that DB2 uses for that table.

About this task

When DB2 prepares an SQL statement that refers to a created temporary table, the statistics that it uses depend on whether the table was already instantiated. If the table was already instantiated, DB2 uses the cardinality and number of pages that are maintained for that table in storage. If the table was never yet instantiated, DB2 uses the CARDF and NPAGES column values of the SYSTABLES row for the created temporary table. These columns normally contain default (-1) values because DB2 utilities cannot collect statistics for created temporary tables.

Procedure

To set statistics for created temporary tables:

Modify the values in the CARDF and NPAGES columns in the row for the created temporary table in the SYSIBM.SYSTABLES catalog table. These values become the default values that are used if more accurate values are not available or cannot be used. The more accurate values are available only for dynamic SQL statements that are prepared after the instantiation of the created temporary table, but within the same unit of work. If the result of the dynamic bind is destined for the dynamic statement cache, these more accurate values are not used.

Related tasks:

Creating created temporary tables (DB2 Administration Guide)

Modifying catalog statistics to influence access path selection

Related reference:

SYSIBM.SYSTABLES table (DB2 SQL)

Deciding whether to rebind after you collect statistics

You do not always need to rebind all applications after you gather statistics.

About this task

You might need to rebind applications after you collect statistics. This action is required only if the access path statistics changed significantly after you last bound the applications and performance suffers as a result. However, it is best to rebind applications during migration DB2 to a new version, to generate new run time structures. You can specify the APREUSE bind option to specify that DB2 generates new run time structures but tries to reuse existing access paths.

Procedure

When performance degrades to an unacceptable level, take any of the following actions:

- Analyze the statistics that are described in the recommendations in this information to help you develop your own guidelines for when to rebind.
- In most cases, rebind applications when the following conditions occur:
  - CLUSTERRATIOF changes to less than 80% (a value of 0.80).
  - NLEAF changes more than 20% from the previous value.
  - NLEVELS changes (only if it was more than a two-level index to begin with).
  - NPAGES changes more than 20% from the previous value.
  - NACTIVEF changes more than 20% from the previous value.
  - The range of HIGH2KEY to LOW2KEY range changes more than 20% from the range previously recorded.
  - Cardinality changes more than 20% from previous range.
  - Distribution statistics change most of the frequent column values, or a new value appears in the frequently occurring values.

Related tasks:

Rebinding an application (DB2 Application programming and SQL)

Monitoring catalog statistics

Related reference:

Statistics used for access path selection

Statistics used for access path selection

DB2 uses statistics from certain catalog table columns when selecting query access paths.

PSPI

DB2 uses certain values from the catalog tables directly when selecting access paths.

For example, the SYSTABLES and SYSTABLESPACE catalog tables indicate how much data the tables referenced by a query contain and how many pages hold data, the SYSINDEXES table indicates the most efficient index for a query, and the SYSCOLUMNS and SYSCOLDIST catalog tables indicate estimated filter factors for predicates.

Important: Use care when issuing SQL statements or using tools to update statistics values in catalog tables. If such updates introduce invalid data, unpredictable results can occur, including abends for RUNSTATS and other utilities. If such problems occur, you can run the RUNSTATS utility and collect statistics at the table space level to resolve the problems, in most cases.

The following tables list columns in catalog tables that DB2 uses for access path selection, values that trigger the use of a default value, and corresponding default values. Catalog table columns that are used directly by DB2 during access path selection are identified by a "Yes" value in the Used for access paths? column of the following tables.

Every table that RUNSTATS updates

As shown in the following table, the STATSTIME column is updated in every table that RUNSTATS updates.

Table 88. Columns in every table that RUNSTATS updates that are used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
STATSTIME	Yes	Yes	No	If updated most recently by RUNSTATS, the date and time of that update, not updatable in SYSINDEXPART and SYSTABLEPART. Used for access path selection for SYSCOLDIST if duplicate column values exist for the same column (by user insertion).

SYSIBM.SYSCOLDIST table (DB2 SQL)

Contains table level frequency, histogram, and multi-column cardinality statistics used by the DB2 to estimate filter factors. The columns in this catalog table that are used for access path selection are shown in the following table.

Table 89. Columns in the SYSIBM.SYSCOLDIST table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDF	Yes	Yes	Yes	For TYPE C, the number of distinct values gathered in the column group; for TYPE F, the number of distinct values for the column group -1; for TYPE='H', the number of distinct values in the column group of the interval indicated by the value of the QUANTILENO column.
COLGROUPCOLNO	Yes	Yes	Yes	The set of columns associated with the statistics. Contains an empty string if NUMCOLUMNS = 1.
COLVALUE	Yes	Yes	Yes	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	Yes	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	No	Yes	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	No	Yes	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
NUMCOLUMNS	Yes	Yes	Yes	The number of columns that are associated with the statistics. The **default value** is 1.
TYPE	Yes	Yes	Yes	The type of statistics gathered: **C** Cardinality **F** Frequent value **N** Non-padded **H** Histogram statistics
QUANTILENO	Yes	No	Yes	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.

SYSIBM.SYSCOLDISTSTATS table (DB2 SQL)

Contains partition-level frequency, histogram, and multi-column cardinality statistics that are used by RUNSTATS to aggregate table-level frequency, histogram, and multi-column cardinality statistics that are stored in SYSIBM.SYSCOLDIST. The columns in this catalog table that are used for access path selection are shown in the following table.

Table 90. Columns in the SYSCOLDISTSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='F' or TYPE='N' the number of rows or keys in the partition for which the FREQUENCYF value applies; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
COLGROUPCOLNO	Yes	Yes	No	The set of columns associated with the statistics.
COLVALUE	Yes	Yes	No	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	No	No	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
KEYCARDDATA	Yes	Yes	No	The internal representation of the estimate of the number of distinct values in the partition.
LOWVALUE	Yes	No	No	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
NUMCOLUMNS	Yes	Yes	No	The number of columns associated with the statistics. The **default value** is 1.
TYPE	Yes	Yes	No	The type of statistics gathered: **C** Cardinality **F** Frequent value **N** Non-padded **H** Histogram statistics
QUANTILENO	Yes	No	No	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.

SYSIBM.SYSCOLSTATS table (DB2 SQL)

Contains partition-level column statistics that are used by DB2 to determine the degree of parallelism, and are also sometimes used to bound filter factor estimates. The columns in this catalog table that are used for access path selection are shown in the following table.

Table 91. Columns in the SYSIBM.SYSCOLSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
COLCARD	Yes	Yes	Yes	The number of distinct values in the partition. Do not update this column manually without first updating COLCARDDATA to a value of length 0. For XML column indicators, NODEID columns, and XML tables, this value of this column is set to -2.
COLCARDDATA	Yes	Yes	No	The internal representation of the estimate of the number of distinct values in the partition. A value appears here only if RUNSTATS TABLESPACE is run on the partition. Otherwise, this column contains a string of length 0, indicating that the actual value is in COLCARD.
HIGHKEY	Yes	Yes	Yes	First 2000 bytes of the highest value of the column within the partition.If the partition is empty, the value is set to a string of length 0. For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.
HIGH2KEY	Yes	Yes	No	First 2000 bytes of the second highest value of the column within the partition. If the partition is empty, the value is set to a string of length 0. For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank. This column is updated with decoded values if the column is a randomized key column.
LOWKEY	Yes	Yes	Yes	First 2000 bytes of the lowest value of the column within the partition. If the partition is empty, the value is set to a string of length 0.For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.
LOW2KEY	Yes	Yes	No	First 2000 bytes of the second lowest value of the column within the partition.If the partition is empty, the value is set to a string of length 0.For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank. This column is updated with decoded values if the column is a randomized key column.
PARTITION	Yes	Yes	Yes	Partition number for the table space that contains the table in which the column is defined.

SYSIBM.SYSCOLUMNS table (DB2 SQL)

The columns in this catalog table that are used for access path selection are shown in the following table.

Table 92. Columns in the SYSCOLUMNS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
COLCARDF	Yes	Yes	Yes	Estimated number of distinct values in the column, -1 to trigger use of the **default value** (25) and -2 for auxiliary indexes, XML column indicators, NODEID columns, and XML tables.
HIGH2KEY	Yes	Yes	Yes	First 2000 bytes of the second highest value in this column.If the table is empty, the value is set to a string of length 0. For auxiliary indexes, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.RUNSTATS does not update HIGH2KEY if the column is a randomized key column.
LOW2KEY	Yes	Yes	Yes	First 2000 bytes of the second lowest value in this column.If the table is empty, the value is set to a string of length 0. For auxiliary indexes, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.RUNSTATS does not update LOW2KEY if the column is a randomized key column.

SYSIBM.SYSINDEXES table (DB2 SQL)

Contains table-level index statistics, that are used by DB2 for index costing. The following columns in this catalog table are used for access path selection.

Table 93. Columns in the SYSINDEXES catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
AVGKEYLEN	Yes	No	No	Average key length.
CLUSTERED	Yes	Yes	No	Whether the table is actually clustered by the index. The value of this column is set to blank for auxiliary indexes, NODEID indexes, and XML indexes.
CLUSTERING	No	No	Yes	Whether the index was created using CLUSTER.

Table 93. Columns in the SYSINDEXES catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CLUSTERRATIOF	Yes	Yes	Yes	A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order. For a partitioned index, it is the weighted average of all index partitions in terms of the number of rows in the partition. The value of this column is set to -2 for auxiliary indexes, NODEID indexes, and XML indexes. If this columns contains the default, 0, DB2 uses the value in CLUSTERRATIO, a percentage, for access path selection.
FIRSTKEYCARDF	Yes	Yes	Yes	Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition, -1 to trigger use of the **default value** (25).
FULLKEYCARDF	Yes	Yes	Yes	Number of distinct values of the full key, -1 to trigger use of the **default value** (25).
NLEAF	Yes	Yes	Yes	Number of active leaf pages in the index, -1 to trigger use of the **default value** (SYSTABLES.CARD/300).
NLEVELS	Yes	Yes	Yes	Number of levels in the index tree, -1 to trigger use of the **default value** (2).
SPACEF	Yes	Yes	No	Disk storage in KB.
DATAREPEATFACTORF	Yes	Yes	Yes	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.

SYSIBM.SYSINDEXPART table (DB2 SQL)

Contains statistics for index space utilization and index organization. For partitioning index of an index controlled partitioned table space, the limit key column is also used in limited partition scan scenarios. The following columns in this catalog table are used for access path selection.

Table 94. Columns in the SYSINDEXPART catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
AVGKEYLEN	Yes	No	No	Average key length.
CARDF	Yes	No	No	Number of rows or LOBs referenced by the index or partition
DSNUM	Yes	Yes	No	Number of data sets.

Table 94. Columns in the SYSINDEXPART catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
EXTENTS	Yes	Yes	No	Number of data set extents (for multiple pieces, the value is for the extents in the last data set).
FAROFFPOSF	Yes	No	No	Number of times that accessing a different, “far-off” page is necessary when accessing all the data records in index order. Each time that DB2 accesses a far-off page, accessing the “next” record in index order probably requires I/O activity. For nonsegmented table spaces, a page is considered far-off from the present page if the two page numbers differ by 16 or more. For segmented table spaces, a page is considered far-off from the present page if the two page numbers differ by SEGSIZE * 2 or more. Together, NEAROFFPOSF and FAROFFPOSF indicate how well the index follows the cluster pattern of the table space. For a clustering index, NEAROFFPOSF and FAROFFPOSF approach a value of 0 as clustering improves. A reorganization should bring them nearer their optimal values; however, if a nonzero FREEPAGE value is specified on the CREATE TABLESPACE statement, the NEAROFFPOSF after reorganization reflects the table on which the index is defined. Do not expect optimal values for non-clustering indexes. The value is -1 if statistics have not been gathered.The value is -2 if the index is a hash index, node ID index, or an XML index.
LEAFDIST	Yes	No	No	100 times the number of pages between successive leaf pages.The value is -2 if the index is a node ID index, AUX index, hash index,or an XML index.
LEAFFAR	Yes	Yes	No	Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan. See “LEAFNEAR and LEAFFAR columns” on page 467 for more information.The value is -2 if the index is a hash index, node ID index, or an XML index.

Table 94. Columns in the SYSINDEXPART catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
LEAFNEAR	Yes	Yes	No	Number of leaf pages located physically near previous leaf pages for successive active leaf pages. See LEAFNEAR and LEAFFAR columns for more information.The value is -2 if the index is a hash index, node ID index, or an XML index.
LIMITKEY	No	No	Yes	The limit key of the partition in an internal format, 0 if the index is not partitioned.
NEAROFFPOSF	Yes	No	No	Number of times that accessing a different, “near-off” page would be necessary when accessing all the data records in index order. Each time that DB2 accesses a near-off page, accessing the “next” record in index order would probably require I/O activity. For more information about NEAROFFPOSF, see the description of FAROFFPOSF. NEAROFFPOSF is incremented if the current indexed row is not on the same or next data page of the previous indexed row, and if the distance between the two data pages does not qualify for FAROFFPOSF. For nonsegmented table spaces, a page is considered near-off from the present page if the difference between the two page numbers is greater than or equal to 2, and less than 16. For segmented table spaces, a page is considered near-off from the present page if the difference between the two page numbers is greater than or equal to 2, and less than SEGSIZE * 2. A nonzero value in the NEAROFFPOSF field after a REORG might be attributed to the number of space map pages that are contained in the segmented table space.The value is -2 if the index is a hash index, node ID index, or an XML index.
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the data set.
PSEUDO_DEL_ENTRIES	Yes	Yes	No	Number of pseudo-deleted keys.
SECQTYI	Yes	No	No	Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0.

Table 94. Columns in the SYSINDEXPART catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
SPACE	Yes	No	No	The number of KB of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces)
SQTY	Yes	No	No	The secondary space allocation in 4 KB blocks for the data set.
SPACEF	Yes	Yes	No	Disk storage in KB.

SYSIBM.SYSINDEXSTATS table (DB2 SQL)

Contains partition-level index statistics that are used by RUNSTATS to aggregate table-level index statistics that are stored in SYSIBM.SYSINDEXES. The following columns in this catalog table are used for access path selection.

Table 95. Columns in the SYSINDEXSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CLUSTERRATIOF	Yes	Yes	No	A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order.
FIRSTKEYCARDF	Yes	Yes	No	Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition.
FULLKEYCARDDATA	Yes	Yes	No	The internal representation of the number of distinct values of the full key.
FULLKEYCARDF	Yes	Yes	No	Number of distinct values of the full key.
KEYCOUNTF	Yes	Yes	No	Number of rows in the partition, -1 to trigger use of the value in KEYCOUNT.
NLEAF	Yes	Yes	No	Number of leaf pages in the index.
NLEVELS	Yes	Yes	No	Number of levels in the index tree.
DATAREPEATFACTORF	Yes	Yes	No	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.

SYSIBM.SYSKEYTARGETS table (DB2 SQL)

Contains table-level frequency, histogram, and multi-column cardinality statistics for column-expression index keys. The values are used by DB2 in filter factor estimation algorithms for matched expressions. The following columns in this catalog table are used for access path selection.

Table 96. Columns in the SYSKEYTARGETS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDF	Yes	No	Yes	Number of distinct values for the key-target. The value of this column is set to -2 for NODEID indexes and XML indexes.
HIGH2KEY	Yes	Yes	Yes	Second highest key value
LOW2KEY	Yes	Yes	Yes	Second lowest key value
STATS_FORMAT	Yes	Yes	Yes	Type of statistics gathered: **blank** No statistics have been collected, or VARCHAR column statistical values are padded **N** Varchar statistical values are not padded

SYSIBM.SYSKEYTARGETSTATS table (DB2 SQL)

Contains partition-level key statistics for keys in column-expression indexes. The values are used by RUNSTATS to aggregate table-level key column-expression statistics. The following columns in this catalog table are used for access path selection.

Table 97. Columns in the SYSKEYTARGETSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
HIGHKEY	Yes	Yes	No	Highest key value
HIGH2KEY	Yes	Yes	No	Second highest key value
LOWKEY	Yes	Yes	No	Lowest key value
LOW2KEY	Yes	Yes	No	Second lowest key value
STATS_FORMAT	Yes	Yes	No	Type of statistics gathered: **blank** No statistics have been collected, or VARCHAR column statistical values are padded **N** Varchar statistical values are not padded

SYSIBM.SYSKEYTGTDIST table (DB2 SQL)

Contains table-level frequency, histogram, and multi-column cardinality statistics for column-expression index keys. The values are used by DB2 in filter factor estimation algorithms for matched expressions. The following columns in this catalog table are used for access path selection.

Table 98. Columns in the SYSKEYTGTDIST catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDF	Yes	Yes	Yes	For TYPE C, Number of distinct values gathered in the key group; for TYPE F, number of distinct values for the key group -1; for TYPE='H', the number of distinct values in the column group of the interval indicated by the value in the QUANTILENO column.
KEYGROUPKEYNO	Yes	Yes	Yes	The set of KEYS associated with the statistics. Contains an empty string if NUMKEYS = 1.
KEYVALUE	Yes	Yes	Yes	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	Yes	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of KEYVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	Yes	Yes	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	Yes	Yes	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
NUMKEYS	Yes	Yes	Yes	The number of keys associated with the statistics. The **default value** is 1.
TYPE	Yes	Yes	Yes	The type of statistics gathered: **C** Cardinality **F** Frequent value **N** Non-padded **H** Histogram statistics
QUANTILENO	Yes	Yes	Yes	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.

SYSIBM.SYSKEYTGTDISTSTATS table (DB2 SQL)

Contains partition-level frequency, histogram, and multi-column cardinality statistics for column-expression index keys. The values are used by RUNSTATS to aggregate table-level statistics that are stored in SYSIBM.SYSCOLDIST. The following columns in this catalog table are used for access path selection.

Table 99. Columns in the SYSKEYTGTDISTSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
KEYVALUE	Yes	Yes	No	The set of keys associated with the statistics
KEYGROUPKEYNO	Yes	Yes	No	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of KEYVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	Yes	No	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	Yes	No	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
QUANTILENO	Yes	Yes	No	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.

SYSIBM.SYSLOBSTATS table (DB2 SQL)

Contains LOB table space statistics. The following columns in this catalog table are used for access path selection.

Table 100. Columns in the SYSLOBSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
AVGSIZE	Yes	Yes	No	Average size of a LOB in bytes.
FREESPACE	Yes	Yes	No	The number of KB of available space in the LOB table space.

Table 100. Columns in the SYSLOBSTATS catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
ORGRATIO	Yes	Yes	No	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized. A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space has an ORGRATIO value of 100.00.

SYSIBM.SYSROUTINES table (DB2 SQL)

Contains statistics for table functions. The following columns in this catalog table are used for access path selection.

Table 101. Columns in the SYSROUTINES catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDINALITY	No	Yes	Yes	The predicted cardinality of a table function, -1 to trigger use of the **default value** (10 000)
INITIAL_INSTS	No	Yes	Yes	Estimated number of instructions executed the first and last time the function is invoked, -1 to trigger use of the **default value** (40 000)
INITIAL_IOS	No	Yes	Yes	Estimated number of IOs performed the first and last time the function is invoked, -1 to trigger use of the **default value** (0)
INSTS_PER_INVOC	No	Yes	Yes	Estimated number of instructions per invocation, -1 to trigger use of the **default value** (4 000)
IOS_PER_INVOC	No	Yes	Yes	Estimated number of IOs per invocation, -1 to trigger use of the **default value** (0)

SYSIBM.SYSTABLEPART table (DB2 SQL)

Contains statistics for space utilization. The following columns in this catalog table are used for access path selection.

Table 102. Columns in the SYSTABLEPART catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
AVGROWLEN	Yes	No	No	Average row length
CARDF	Yes	No	No	Total number of rows in the table space or partition. For LOB table spaces, the number of LOBs in the table space.

Table 102. Columns in the SYSTABLEPART catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
DSNUM	Yes	Yes	No	Number of data sets.
EXTENTS	Yes	Yes	No	Number of data set extents (for multiple pieces, the value is for the extents in the last data set).
FARINDREF	Yes	No	No	Number of rows that are relocated far from their original page. If an update operation increases the length of a record by more than the amount of available space in the page in which it is stored, the record is moved to another page. Until the table space is reorganized, the record requires an additional page reference when it is accessed. The sum of NEARINDREF and FARINDREF is the total number of such records. For nonsegmented table spaces, a page is considered "near" the present page if the two page numbers differ by 16 or fewer; otherwise, it is "far from" the present page. For segmented table spaces, a page is considered "near" the present page if the two page numbers differ by (SEGSIZE * 2) or less. Otherwise, it is "far from" its original page. A record that is relocated near its original page tends to be accessed more quickly than one that is relocated far from its original page.
NEARINDREF	Yes	No	No	Number of rows relocated near their original page.

Table 102. Columns in the SYSTABLEPART catalog table that are updated by RUNSTATS or used for access path selection (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
PAGESAVE	Yes	No	No	Percentage of pages, times 100, saved in the table space or partition as a result of using data compression. For example, a value of 25 indicates a savings of 25%, so that the required pages are only 75% of what would be required without data compression. The value is 0 if no savings from using data compression are likely, or if statistics have not been gathered. The value can be negative if using data compression causes an increase in the number of pages in the data set. This calculation includes the overhead bytes for each row, the required bytes for the dictionary, and the required bytes for the current FREEPAGE and PCTFREE specification for the table space and partition. This calculation is based on an average row length, and the result varies depending on the actual lengths of the rows.
PERCACTIVE	Yes	No	No	Percentage of space occupied by active rows, containing actual data from active tables, -2 for LOB table spaces. This value is influenced by the PCTFREE and the FREEPAGE parameters on the CREATE TABLESPACE statement and by unused segments of segmented table spaces.
PERCDROP	Yes	No	No	For non-segmented table spaces, the percentage of space occupied by rows of data from dropped tables; for segmented table spaces, 0.
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the data set.
SECQTYI	Yes	No	No	Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0.
SPACE	Yes	No	No	The number of KB of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces).
SPACEF	Yes	Yes	No	Disk storage in KB.
SQTY	Yes	No	No	The secondary space allocation in 4K blocks for the data set

SYSIBM.SYSTABLES table (DB2 SQL)

Contains table-level table statistics that are used by DB2 throughout the query costing process. The following columns in this catalog table are used for access path selection.

Table 103. Columns in the SYSTABLES catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
AVGROWLEN	Yes	Yes	No	Average row length of the table specified in the table space.
CARDF	Yes	Yes	Yes	Total number of rows in the table or total number of LOBs in an auxiliary table, -1 to trigger use of the **default value** (10 000).
EDPROC	No	No	Yes	Non-blank value if an edit exit routine is used.
NPAGES	Yes	Yes	Yes	Total number of pages on which rows of this table appear, -1 to trigger use of the **default value** (CEILING(1 + CARD/20))
NPAGESF	Yes	Yes	Yes	Number of pages used by the table.
PCTPAGES	Yes	Yes	No	For non-segmented table spaces, percentage of total pages of the table space that contain rows of the table; for segmented table spaces, the percentage of total pages in the set of segments assigned to the table that contain rows of the table.
PCTROWCOMP	Yes	Yes	Yes	Percentage of rows compressed within the total number of active rows in the table.
SPACEF	Yes	Yes	No	Disk storage in KB.

SYSIBM.SYSTABLESPACE table (DB2 SQL)

Contains table-space level statistics that are used by DB2 for costing of non-segmented table spaces. The following columns in this catalog table are used for access path selection.

Table 104. Columns in the SYSTABLESPACE catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
AVGROWLEN	Yes	No	No	Average row length.
NACTIVEF	Yes	Yes	Yes	Number of active pages in the table space, the number of pages touched if a cursor is used to scan the entire file, 0 to trigger use of the value in the NACTIVE column instead. If NACTIVE contains 0, DB2 uses the **default value** (CEILING(1 + CARD/20)).
SPACE	Yes	No	No	Disk storage in KB.
SPACEF	Yes	Yes	No	Disk storage in KB.

SYSIBM.SYSTABSTATS table (DB2 SQL)

Contains partition-level table statistics that are used by DB2 when costing limited partition scans, and are also used by RUNSTATS to aggregate table-level table statistics that are stored in SYSIBM.SYSTABLES. The following columns in this catalog table are used for access path selection.

Table 105. Columns in the SYSTABSTATS catalog table that are updated by RUNSTATS or used for access path selection

Column name	Set by RUNSTATS?	User can update?	Used for access paths? [1]	Description
CARDF	Yes	Yes	Yes	Total number of rows in the partition, -1 to trigger use of the value in the CARD column. If CARD is -1, DB2 uses a **default value**(10 000).
NACTIVE	Yes	Yes	No	Number of active pages in the partition.
NPAGES	Yes	Yes	Yes	Total number of pages on which rows of the partition appear, -1 to trigger use of the **default value** (CEILING(1 + CARD/20)).
PCTPAGES	Yes	Yes	No	Percentage of total active pages in the partition that contain rows of the table.
PCTROWCOMP	Yes	Yes	No	Percentage of rows compressed within the total number of active rows in the partition, -1 to trigger use of the **default value** (0).

Notes:

1. Statistics on LOB-related values are not used for access path selection. The SYSCOLDISTSTATS and SYSINDEXSTATS catalog tables are not used for parallelism access paths. Information in the SYSCOLSTATS catalog table (the CARD, HIGHKEY, LOWKEY, HIGH2KEY, and LOW2KEY columns) is used information is used to determine the degree of parallelism.

PSPI

Related tasks:

Investigating access path problems

Related reference:

DB2 catalog tables (DB2 SQL)

LOAD (DB2 Utilities)

RUNSTATS (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

REORG INDEX (DB2 Utilities)

How clustering affects access path selection

Whether and how your data is clustered affects how DB2 chooses an access path. The value of the CLUSTERRATIOF column gives an indication of how closely the order of the index entries on the index leaf pages matches the actual ordering of the rows on the data pages.

In general, the closer that the value of the CLUSTERRATIOF column is to 100%, the more closely the ordering of the index entries matches the actual ordering of the rows on the data pages. The actual formula is quite complex and accounts for indexes with many duplicates; in general, for a given index, the more duplicates, the higher the CLUSTERRATIOF value.

Here are some things to remember about the effect of the CLUSTERRATIOF column on access paths:

- CLUSTERRATIOF is an important input to the cost estimates that are used to determine whether an index is used for an access path, and, if so, which index to use.
- If the access is INDEXONLY, then this value does not apply.
- The higher the CLUSTERRATIOF value, the lower the cost of referencing data pages during an index scan is.
- For an index that has a CLUSTERRATIOF less than 80%, sequential prefetch is not used to access the data pages.
- A slight reduction in CLUSTERRATIOF for a table with a large number of rows can represent a much more significant number of unclustered rows than for a table with a small number of rows.For example, A CLUSTERRATIOF of 99% for a table with 100,000,000 rows represents 100,000 unclustered rows. Whereas, the CLUSTERRATIOF of 95% for a table with 100,000 rows represents 5000 unclustered rows.
- For indexes that contain either many duplicate key values or key values that are highly clustered in reverse order, cost estimation that is based purely on CLUSTERRATIOF can lead to repetitive index scans. In the worst case, an entire page could be scanned one time for each row in the page. DB2 access path selection can avoid this performance problem by using a cost estimation formula based on the DATAREPEATFACTORF statistic to choose indexes. Whether DB2 will use this statistic depends on the value of the STATCLUS subsystem parameter.

The following figures below show the comparison between an index scan on an index with a high cluster ratio and an index with a lower cluster ratio.

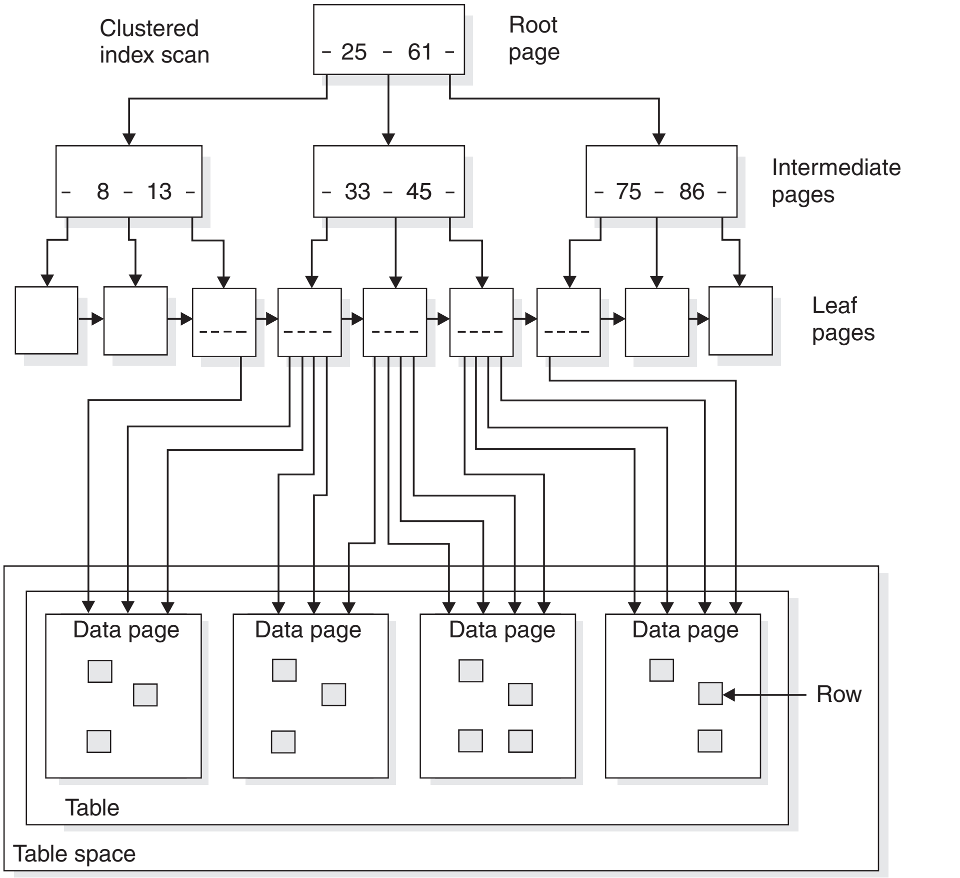


Figure 35. A clustered index scan. This figure assumes that the index is 100% clustered.

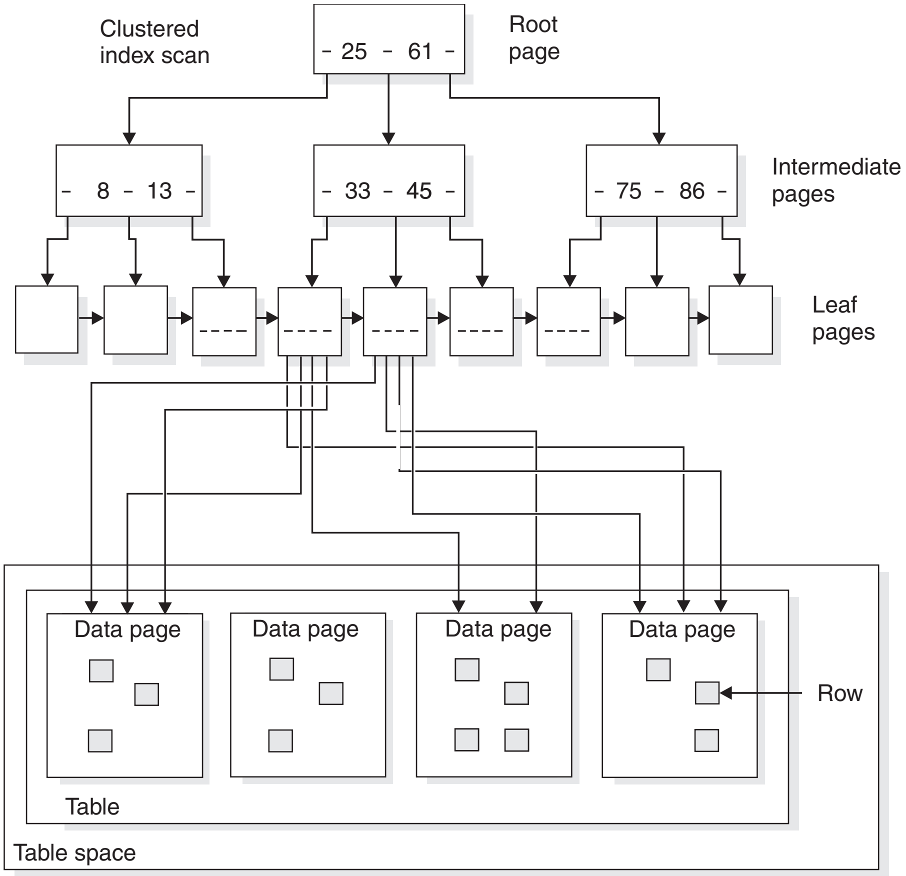


Figure 36. A nonclustered index scan. In some cases, DB2 can access the data pages in order even when a nonclustered index is used.

Related tasks:

Maintaining data organization

Determining when to reorganize indexes

Related reference:

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

SYSIBM.SYSINDEXES table (DB2 SQL)

Additional statistics that provide index costs

Certain statistics in the SYSINDEXES catalog table also provide information about costs associated with index processing.

PSPI

The following columns of the SYSIBM.SYSINDEXES catalog table provide cost information for index processing:

FIRSTKEYCARDF
The number of distinct values of the first index key column. When an indexable equal predicate is specified on the first index key column, 1/FIRSTKEYCARDF is the filter factor for the predicate and the index. The higher the number is, the less the cost is.

FULLKEYCARDF

The number of distinct values for the entire index key. When indexable equal predicates are specified on all the index key columns, 1/FULLKEYCARDF is the filter factor for the predicates and the index. The higher the number is, the less the cost is.

When the number of matching columns is greater than 1 and less than the number of index key columns, the filtering of the index is located between 1/FIRSTKEYCARDF and 1/FULLKEYCARDF.

NLEAF

The number of active leaf pages in the index. This value also includes the number of pseudo-empty pages in the index. NLEAF is a portion of the cost to scan the index. The smaller the number is, the less the cost is. It is also less when the filtering of the index is high, which comes from FIRSTKEYCARDF, FULLKEYCARDF, and other indexable predicates.

NLEVELS

The number of levels in the index tree. NLEVELS is another portion of the cost to traverse the index. The same conditions as NLEAF apply. The smaller the number is, the less the cost is.

DATAREPEATFACTORF

The number of times that data pages are repeatedly scanned after the index key is ordered.

PSPI

Dynamic collection of index filtering estimates

When certain statistics are unavailable, DB2 might access non-leaf index pages to obtain accurate index filtering estimates and improve access path selection.

When an SQL statement meets the following criteria, DB2 might obtain statistics for access path selection from non-leaf index pages:

- The SQL statement contains local predicates that match for index access.
- The predicates contain literal values, or the one of the REOPT(AUTO), (ONCE), or (ALWAYS) bind options is specified.

DB2 records information about the collected statistics in the DSN_COLDIST_TABLE and DSN_KEYTGTDIST_TABLE tables.

The following conditions might trigger the collection of statistics during optimization:

- Statistics indicate that the matching index key range is outside the range of the LOW2KEY and HIGH2KEY values.
- Histogram statistics indicate that the matching index access-key qualifies no rows.
- Frequency statistics indicate that the matching index access-key qualifies no rows.
- Statistics indicate that one or more qualified partitions contain no rows.
- Statistics indicate that the table contains no rows.
- The table is defined with the VOLATILE option, or passes the threshold specified by the NPGTHRSH subsystem parameter.

The only way to prevent the collection of these statistics is to ensure that the preceding conditions the are not met. To do this you might need to run the RUNSTATS utility or collect inline statistics with another unitility to update missing or out-of-date statistics or alter the table so that it is no longer VOLATILE.

DB2 might not always access the index even when the preceding conditions are true. For example, in the case of a fully matched unique index, such that only one row qualifies for the matching predicate, DB2 does not obtain statistics from the index.

Related tasks:

Reoptimizing SQL statements at run time

Favoring index access

Related reference:

DSN_COLDIST_TABLE

DSN_KEYTGTDIST_TABLE

NPGTHRSH in macro DSN6SPRM (DB2 Installation and Migration)

ALTER TABLE (DB2 SQL)

Chapter 35. Setting up your system for real-time statistics

You can use *real-time statistics* to determine when objects require maintenance by utilities such as REORG, RUNSTATS, or COPY. These statistics can also be used to automate the scheduling of such utility jobs.

About this task

DB2 always generates in-memory statistics for each table space and index space in your system, including catalog objects. DB2 periodically writes these *real-time statistics* to certain catalog tables, at a specified interval. You can use these real-time statistics to determine when objects require maintenance by utilities such as REORG, RUNSTATS, or COPY. These statistics can also be used to automate the scheduling of such utility jobs.

For partitioned spaces, DB2 generates information for each partition. However, no statistics are generated for the SYSRTSTS table space and its corresponding index spaces. Similarly DB2 does not generate statistics for certain items in the directory, such as the SYSLGRNX table space and its corresponding indexes DSNLLX01 and DSNLLX02, for example.

Procedure

1. Set the interval for writing real-time statistics to the tables. You can modify STATSINT subsystem parameter to set the interval. The default interval is 30 minutes.
2. Run the appropriate utilities to establish baseline values for the real-time statistics. Many columns in the real-time statistics tables show the number of times that an operation occurred between the last time a particular utility was run and the time when the real-time statistics are written. Therefore, starting values are needed. For example, STATSINSERT in SYSTABLESPACESTATS indicates the number of records or LOBs that have been inserted after the last RUNSTATS utility was run on the table space or partition.

   For migration from DB2 Version 8, if you did not use real-time statistics in DB2 Version 8, some real-time statistics values remain null in DB2 10 until you run the appropriate utility to establish starting points for the counters.

Related reference:

How utilities affect the real-time statistics

REAL TIME STATS field (STATSINT subsystem parameter) (DB2 Installation and Migration)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

DSNACCOX stored procedure (DB2 SQL)

When DB2 externalizes real-time statistics

In-memory statistics for DB2 objects are written to DB2 catalog tables at a specified interval, and in other certain situations.

DB2 uses an asynchronous task to externalize the in-memory statistics to the real-time statistics tables. To accomplish this task, DB2 completes the following actions:

1. Examines the in-memory statistics.
2. Calculates the new total values.
3. Updates the real-time statistics tables.
4. Resets the in-memory statistics

DB2 externalizes real-time statistics in the following situations:

- At the end of the time interval that you specify during installation in the value of the STATSINT subsystem parameter.
- During utility operations. Some utilities modify the statistics tables.
- When you issue the ACCESS DATABASE command and specify the MODE(STATS) option, DB2 externalizes real-time statistics for the specified objects. You can use this method to externalize the real-time statistics immediately, before your invoke processes that depend on the accuracy of the real-time statistics values, such as the DSNACCOX stored procedure.
- When you issue START DATABASE or STOP DATABASE commands. These commands externalize statistics only for the databases and table spaces that are specified by the commands. No statistics are externalized when the DSNDB06 database is stopped.
- When you issue the STOP DB2 command. DB2 writes any statistics that are in memory to the statistics tables when you issue this command.

DB2 does not externalize real-time statistics in the following situations:

- When DB2 is started with the ACCESS(MAINT) option and the DEFER ALL option is in effect. In this case, all statistics changes are lost.
- When objects are in a UTUT, UTRO, or UTRW state.
- At a tracker site.

Because DB2 holds locks when it externalizes real-time statistics, timeout and deadlock situations are possible when two processes attempt to externalize real-time statistics simultaneously. For example, such timeouts or deadlocks might occur if you issue a STOP DATABASE command when DB2 is already externalizing real-time statistics at the end of the interval specified by the STATSINT subsystem parameter.

Related concepts:

How DB2 maintains in-memory statistics in data sharing (DB2 Data Sharing Planning and Administration)

Lock contention

Related reference:

REAL TIME STATS field (STATSINT subsystem parameter) (DB2 Installation and Migration)

-STOP DATABASE (DB2) (DB2 Commands)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

Updating real-time statistics immediately

You can issue a command to externalize in-memory statistics to the real-time statistics catalog table immediately.

About this task

Normally, in-memory statistics are externalized to the real-time statistics table only when the interval specified by the STATSINT subsystem parameter elapses. The result can be problematic when operations, such as DSNACCOX stored procedure, that depend upon accurate statistics are invoked long after that last interval period, especially for objects that are subject to frequent changes.

Procedure

To immediately externalize in-memory statistics to the real-time statistics tables:

Issue the ACCESS DATABASE command, and specify the MODE(STATS) option. For example, you might issue the following command to externalize all in-memory statistics for the subsystem to the realtime statistics tables:

```
ACCESS DB(*) SP(*) MODE(STATS)
```

Although ranges and partial asterisk values, such as `DB(A:B)` or `DB(A*)`, are supported for the DB or SP options, the results are inefficient. Therefore, the recommendation is to avoid specifying range or partial asterisks for these values whenever possible, usually by specifying to the next higher object level instead. The following table shows the recommended combinations of values for different object levels:

Option	Description
Partition-level statistics	ACCESS DB(*db-name*) SP (sp-name) MODE (STATS) PART (*partition-name*)
Tablespace level statistics	ACCESS DB(*db-name*) SP (sp-name) MODE (STATS)
Database-level statistics	ACCESS DB(*db-name*) SP (*) MODE (STATS)
Subsystem-level statistics	ACCESS DB(*) SP (*) MODE (STATS)

Related concepts:

When DB2 externalizes real-time statistics

Related reference:

-ACCESS DATABASE (DB2) (DB2 Commands)

DSNACCOX stored procedure (DB2 SQL)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

REAL TIME STATS field (STATSINT subsystem parameter) (DB2 Installation and Migration)

How SQL operations affect real-time statistics counters

SQL operations affect the counter columns in the real-time statistics tables. These are the columns that record the number of insert, delete, or update operations, as well as the total counters, TOTALROWS, and TOTALENTRIES.

UPDATE
When you issue an UPDATE statement, DB2 increments the update counters.

INSERT
When you issue an INSERT statement, DB2 increments the insert counters. DB2 keeps separate counters for clustered and unclustered INSERTs.

DELETE
When you issue a DELETE statement, DB2 increments the delete counters.

ROLLBACK
When you issue ROLLBACK statement, or when an implicit rollback occurs, DB2 increments the counters, depending on the type of SQL operation that is rolled back:

Rolled-back SQL statement	Incremented counters
UPDATE	Update counters
INSERT	Delete counters
DELETE	Insert counters

Notice that for INSERT and DELETE, the counter for the inverse operation is incremented. For example, if two INSERT statements are rolled back, the delete counter is incremented by 2.

UPDATE of partitioning keys
If an update to a partitioning key causes rows to move to a new partition, the following real-time statistics are impacted:

Action	Incremented counters
When UPDATE is executed	Update count of old partition = +1 Insert count of new partition = +1
When UPDATE is committed	Delete count of old partition = +1
When UPDATE is rolled back	Update count of old partition = +1 (compensation log record) Delete count of new partition = +1 (remove inserted record)

If an update to a partitioning key does not cause rows to move to a new partition, the counts are accumulated as expected:

Action	Incremented counters
When UPDATE is executed	Update count of current partition = +1 NEAR/FAR indirect reference count = +1 (if overflow occurred)
When UPDATE is rolled back	Update count of current partition = +1 (compensation log record)

Related concepts:

How DB2 rolls back work (DB2 Administration Guide)

Related reference:

DELETE (DB2 SQL)

INSERT (DB2 SQL)

ROLLBACK (DB2 SQL)

UPDATE (DB2 SQL)

How utilities affect the real-time statistics

In general, SQL INSERT, UPDATE, and DELETE statements cause DB2 to modify the real-time statistics. However, certain DB2 utilities also affect the statistics.

For migration from DB2 Version 8, if you did not use real-time statistics in DB2 Version 8, some real-time statistics values remain null in DB2 10 until you run the appropriate utility to establish starting points for the counters.

Related concepts:

How SQL operations affect real-time statistics counters

How DB2 maintains in-memory statistics in data sharing (DB2 Data Sharing Planning and Administration)

How LOAD affects real-time statistics

When you run LOAD REPLACE on a table space or table space partition, you change the statistics associated with that table space or partition.

The table below shows how running LOAD REPLACE on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

Table 106. Changed SYSTABLESPACESTATS values during LOAD REPLACE

Column name	Settings for LOAD REPLACE after RELOAD phase
TOTALROWS	Number of loaded rows or LOBs[1]
DATASIZE	Actual value
NPAGES	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGUPDATES	0
REORGDISORGLOB	0
REORGUNCLUSTINS	0
REORGMASSDELETE	0
REORGNEARINDREF	0
REORGFARINDREF	0
STATSLASTTIME	Current timestamp[2]

Table 106. Changed SYSTABLESPACESTATS values during LOAD REPLACE (continued)

Column name	Settings for LOAD REPLACE after RELOAD phase
STATSINSERTS	0[2]
STATSDELETES	0[2]
STATSUPDATES	0[2]
STATSMASSDELETE	0[2]
COPYLASTTIME	Current timestamp[3]
COPYUPDATEDPAGES	0[3]
COPYCHANGES	0[3]
COPYUPDATELRSN	Null[3]
COPYUPDATETIME	Null[3]

Notes:

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

The table below shows how running LOAD REPLACE affects the SYSINDEXSPACESTATS statistics for an index space or physical index partition.

Table 107. Changed SYSINDEXSPACESTATS values during LOAD REPLACE

Column name	Settings for LOAD REPLACE after BUILD phase
TOTALENTRIES	Number of index entries added[1]
NLEAF	Actual value
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASSDELETE	0
REORGLEAFNEAR	0
REORGLEAFFAR	0
REORGNUMLEVELS	0
STATSLASTTIME	Current timestamp[2]
STATSINSERTS	0[2]
STATSDELETES	0[2]

Table 107. Changed SYSINDEXSPACESTATS values during LOAD REPLACE (continued)

Column name	Settings for LOAD REPLACE after BUILD phase
STATSMASSDELETE	0[2]
COPYLASTTIME	Current timestamp[3]
COPYUPDATEDPAGES	0[3]
COPYCHANGES	0[3]
COPYUPDATELRSN	Null[3]
COPYUPDATETIME	Null[3]

Notes:

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

For a logical index partition:

- A LOAD operation without the REPLACE option behaves similar to a SQL INSERT operation in that the number of records loaded are counted in the incremental counters such as REORGINSERTS, REORGAPPENDINSERT, STATSINSERTS, and COPYCHANGES. A LOAD operation without the REPLACE option affects the organization of the data and can be a trigger to run REORG, RUNSTATS or COPY.
- DB2 does not reset the nonpartitioned index when it does a LOAD REPLACE on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates LOADRLASTTIME when the entire nonpartitioned index is replaced.
- When DB2 does a LOAD RESUME YES on a partition, after the BUILD phase, DB2 increments TOTALENTRIES by the number of index entries that were inserted during the BUILD phase.

Related reference:

LOAD (DB2 Utilities)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

How REORG affects real-time statistics

When you run the REORG utility DB2 modifies some of the real-time statistics for the involved table space or index.

The table below shows how running REORG on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

Table 108. Changed SYSTABLESPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALROWS	Number of rows loaded[1]	For SHRLEVEL REFERENCE: Number of loaded rows or LOBs during RELOAD phase For SHRLEVEL CHANGE: Number of loaded rows or LOBs during RELOAD phase plus number of rows inserted during LOG APPLY phase minus number of rows deleted during LOG phase
DATASIZE	Actual value	Actual value
NPAGES	Actual value	Actual value
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value[3]
REORGDELETES	0	Actual value[3]
REORGUPDATES	0	Actual value[3]
REORGDISORGLOB	0	Actual value[3]
REORGUNCLUSTINS	0	Actual value[3]
REORGMASSDELETE	0	Actual value[3]
REORGNEARINDREF	0	Actual value[3]
REORGFARINDREF	0	Actual value[3]
STATSLASTTIME	Current timestamp[4]	Current timestamp[4]
STATSINSERTS	0[4]	Actual value[3]
STATSDELETES	0[4]	Actual value[3]
STATSUPDATES	0[4]	Actual value[3]
STATSMASSDELETE	0[4]	Actual value[3]
COPYLASTTIME	Current timestamp[5]	Current timestamp
COPYUPDATEDPAGES	0[5]	Actual value[6]
COPYCHANGES	0[5]	Actual value[6]
COPYUPDATELRSN	Null[5]	Actual value[6]
COPYUPDATETIME	Null[5]	Actual value[6]

The table below shows how running REORG affects the SYSINDEXSPACESTATS statistics for an index space or physical index partition.

Table 109. Changed SYSINDEXSPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALENTRIES	Number of index entries added[2]	For SHRLEVEL REFERENCE: Number of added index entries during BUILD phase For SHRLEVEL CHANGE: Number of added index entries during BUILD phase plus number of added index entries during LOG phase minus number of deleted index entries during LOG phase
NLEAF	Actual value	Actual value
NLEVELS	Actual value	Actual value
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value[3]
REORGDELETES	0	Actual value[3]
REORGAPPENDINSERT	0	Actual value[3]
REORGPSEUDODELETES	0	Actual value[3]
REORGMASSDELETE	0	Actual value[3]
REORGLEAFNEAR	0	Actual value[3]
REORGLEAFFAR	0	Actual value[3]
REORGNUMLEVELS	0	Actual value[3]
STATSLASTTIME	Current timestamp[4]	Current timestamp[4]
STATSINSERTS	0[4]	Actual value[3]
STATSDELETES	0[4]	Actual value[3]
STATSMASSDELETE	0[4]	Actual value[3]
COPYLASTTIME	Current timestamp[5]	Unchanged[7]
COPYUPDATEDPAGES	0[5]	Unchanged[7]
COPYCHANGES	0[5]	Unchanged[7]
COPYUPDATELRSN	Null[5]	Unchanged[7]
COPYUPDATETIME	Null[5]	Unchanged[7]

For a logical index partition, DB2 does not reset the nonpartitioned index when it does a REORG on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters and REORGLASTTIME are relative to the last time the entire nonpartitioned index is reorganized. In addition, the REORG counters might be low because, due to the methodology, some index entries are changed during REORG of a partition.

Notes for the preceding tables:

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this

value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.

2. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
3. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
4. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
5. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
6. This is the LRSN or timestamp for the first update that is due to applying the log to the shadow copy.
7. Inline COPY is not supported for SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

Related reference:

REORG TABLESPACE (DB2 Utilities)

REORG INDEX (DB2 Utilities)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

How REBUILD INDEX affects real-time statistics

Rebuilding an index has certain effects on the statistics for the index involved.

The table below shows how running REBUILD INDEX affects the SYSINDEXSPACESTATS statistics for an index space or physical index partition.

Table 110. Changed SYSINDEXSPACESTATS values during REBUILD INDEX

Column name	Settings after BUILD phase
TOTALENTRIES	Number of index entries added[1]
NLEAF	Actual value
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
REBUILDLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASSDELETE	0
REORGLEAFNEAR	0
REORGLEAFFAR	0
REORGNUMLEVELS	0

Notes:

1. Under certain conditions, such as a utility restart, the REBUILD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.

For a logical index partition, DB2 does not collect TOTALENTRIES statistics for the entire nonpartitioned index when it runs REBUILD INDEX. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates REBUILDLASTTIME when the entire nonpartitioned index is rebuilt.

Related reference:

REBUILD INDEX (DB2 Utilities)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

How RUNSTATS affects real-time statistics

When the RUNSTATS job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables.

Only RUNSTATS UPDATE ALL affects the real-time statistics.

The table below shows how running RUNSTATS UPDATE ALL on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

Table 111. Changed SYSTABLESPACESTATS values during RUNSTATS UPDATE ALL

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp[1]	Timestamp of the start of RUNSTATS phase
STATSINSERTS	Actual value[1]	Actual value[2]
STATSDELETES	Actual value[1]	Actual value[2]
STATSUPDATES	Actual value[1]	Actual value[2]
STATSMASSDELETE	Actual value[1]	Actual value[2]
TOTALROWS	Actual value[1]	Actual value[3]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. The value is updated only when SHRLEVEL REFERENCE without sampling is specified.

The table below shows how running RUNSTATS UPDATE ALL on an index affects the SYSINDEXSPACESTATS statistics.

Table 112. Changed SYSINDEXSPACESTATS values during RUNSTATS UPDATE ALL

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp[1]	Timestamp of the start of RUNSTATS phase
STATSINSERTS	Actual value[1]	Actual value[2]

Table 112. Changed SYSINDEXSPACESTATS values during RUNSTATS UPDATE ALL (continued)

Column name	During UTILINIT phase	After RUNSTATS phase
STATSDELETES	Actual value[1]	Actual value[2]
STATSMASSDELETE	Actual value[1]	Actual value[2]
TOTALENTRIES	Actual value[1]	Actual value[3]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. The value is updated only when SHRLEVEL REFERENCE without sampling is specified.

Related reference:

RUNSTATS (DB2 Utilities)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

How COPY affects real-time statistics

When a COPY job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables. Statistics are gathered for a full image copy or an incremental copy, but not for a data set copy.

The table below shows how running COPY on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

Table 113. Changed SYSTABLESPACESTATS values during COPY

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp[1]	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value[1]	Actual value[2]
COPYCHANGES	Actual value[1]	Actual value[2]
COPYUPDATELRSN	Actual value[1]	Actual value[3]
COPYUPDATETIME	Actual value[1]	Actual value[3]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

The table below shows how running COPY on an index affects the SYSINDEXSPACESTATS statistics.

Table 114. Changed SYSINDEXSPACESTATS values during COPY

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp[1]	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value[1]	Actual value[2]
COPYCHANGES	Actual value[1]	Actual value[2]
COPYUPDATELRSN	Actual value[1]	Actual value[3]
COPYUPDATETIME	Actual value[1]	Actual value[3]

Notes:

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

Related reference:

COPY (DB2 Utilities)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

How RECOVER affects real-time statistics

After recovery to the current state, the in-memory counter fields are still valid, so DB2 does not modify them. However, after a point-in-time recovery, the statistics might not be valid.

Consequently, DB2 sets all the REORG, STATS, and COPY counter statistics to null after a point-in-time recovery. After recovery to the current state, DB2 sets NACTIVE, SPACE, and EXTENTS to their new values. After a point-in-time recovery, DB2 sets NLEVELS, NACTIVE, SPACE, and EXTENTS to their new values.

Related reference:

RECOVER (DB2 Utilities)

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

Preventing inaccurate real-time statistics from non-DB2 utility operations

You can prevent non-DB2 utility operations from creating inaccurate real-time statistics.

About this task

Non-DB2 utilities do not affect real-time statistics. Therefore, an object that is the target of a non-DB2 COPY, LOAD, REBUILD, REORG, or RUNSTATS job can cause incorrect statistics to be inserted in the real-time statistics tables.

Procedure

To prevent non-DB2 utilities from creating inaccurate real-time statistics:

1. Stop the table space or index that is the target of the utility operation. his action causes DB2 to write the in-memory statistics to the real-time statistics tables and initialize the in-memory counters. If DB2 cannot externalize the statistics, the STOP command does not fail.
2. Invoke the utility operation.
3. When the utility completes, update the statistics tables with new totals and timestamps, and insert zero values in the incremental counters.

How creating objects affects real-time statistics

When you create a table space or index, DB2 adds statistics to the real-time statistics tables.

A row is inserted into the real-time statistics when a table space or index is created. The time stamp for the creation of the object is stored in the REORGLASTTIME field for the SYSIBM.SYSTABLESPACESTATS catalog table. The values for all fields that contain incremental statistics in the row are set to 0, and real-time statistics for the object are maintained from that time forward.

Related reference:

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

How dropping objects affects real-time statistics

If you drop a table space or index, DB2 deletes its statistics from the real-time statistics tables.

However, if the real-time statistics database is not available when you drop a table space or index, the statistics remain in the real-time statistics tables, even though the corresponding object no longer exists. You need to use SQL DELETE statements to manually remove those rows from the real-time statistics tables.

If a row still exists in the real-time statistics tables for a dropped table space or index, and if you create a new object with the same DBID and PSID as the dropped object, DB2 reinitializes the row before it updates any values in that row.

Real-time statistics for special objects

Objects with certain special characteristics have special implications for real-time statistics.

The following types of objects have specific implications for real-time statistics:

Objects in work file databases

Although you cannot run utilities on objects in the work files databases, DB2 records the NACTIVE, SPACE, and EXTENTS statistics on table spaces in those databases.

DB2 can use partition-by-growth table spaces in work file databases. If you choose to use a partition-by-growth table space in a work file database, you might need to update your real-time statistics programs to work with partition-by-growth statistics. The SYSIBM.SYSTABLESPACESTATS table maintains information about disk storage allocation. This table creates one row for each partition of partition-by-growth table spaces that reside in the WORKFILE database.

DEFINE NO objects
For objects that are created with DEFINE NO, no row is inserted into the real-time statistics table until the object is physically defined.

Read-only or unmodified objects
DB2 does not externalize the NACTIVE, SPACE, or EXTENTS statistics for read-only objects or objects that are not modified

How the EXCHANGE command affects real-time statistics

When the EXCHANGE command is used for clone tables real-time statistics are affected.

The values of the INSTANCE columns in the SYSTABLESPACESTATS and SYSINDEXSPACESTATS tables identify the VSAM data set that is associated with the real-time statistics row. For a cloned object, the real-time statistics table row might contain two rows, one for each instance of the object.

Utility operations and SQL operations can be run separately on each instance of the cloned object. Therefore, each instance of an object can be used to monitor activity, and allow recommendations on whether the base, the clone, or both objects require a REORG, RUNSTATS, or COPY operation.

How real-time statistics affect sort work data set allocation for DB2 utilities

DB2 uses real-time statistics data to estimate the size of sort work data sets to allocate for certain utilities.

If real-time statistics data is available, DB2 uses real-time statistics to determine data set sizes for dynamically allocated sort work data sets for the following utilities:

- CHECK INDEX
- REBUILD INDEX
- REORG TABLESPACE
- RUNSTATS with the COLGROUP keyword

Related concepts:

How real-time statistics are used by DB2 utilities (DB2 Utilities)

Related reference:

UT SORT DATA SET ALLOCATION field (UTSORTAL subsystem parameter) (DB2 Installation and Migration)

CHECK INDEX (DB2 Utilities)

REBUILD INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

RUNSTATS (DB2 Utilities)

Improving concurrency for real-time statistics data

You can specify certain options to prevent timeouts and deadlocks when you work with data in real-time statistics tables.

Procedure

To reduce the risk of timeouts and deadlocks:

- When you run COPY, RUNSTATS, or REORG on the real-time statistics objects, use SHRLEVEL CHANGE.
- When you execute SQL statements to query the real-time statistics tables, use uncommitted read isolation.

Related tasks:

Improving concurrency

Related reference:

COPY (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

REORG INDEX (DB2 Utilities)

RUNSTATS (DB2 Utilities)

Recovering the real-time statistics tables

When you recover a DB2 subsystem after a disaster, DB2 starts with the ACCESS(MAINT) option. No statistics are externalized in this state.

Procedure

Consequently, you need to perform the following actions on the real-time statistics database:

- Recover the real-time statistics objects after you recover the DB2 catalog and directory.
- Start the real-time statistics database explicitly, after DB2 restart.

Accuracy of real-time statistics

In general, the real-time statistics values are very accurate. However, several factors can affect the accuracy of the statistics.

The following factors might affect the accuracy of real-time statistics values:

- Certain utilities not run after migration from DB2 Version 8 to DB2 10. Many real-time statistics values count the number of times something happened since the last time that a particular utility was run. If you did not use real-time statistics in DB2 Version 8, these values remain null in DB2 10 until you establish a starting point by running the appropriate utility.
- Certain utility restart scenarios.
- A utility failure that leaves the object in a utility-in-progress state (UTRW, UTRO, or UTUT).
- Certain utility operations that leave indexes in a database restrictive state, such as RECOVER-pending (RECP). Always consider the database restrictive state of objects before accepting a utility recommendation that is based on real-time statistics.

- Certain third-party vendor solutions that do not correctly manage real-time statistics.
- A DB2 subsystem failure.
- A notify failure in a data sharing environment.

If you think that some statistics values might be inaccurate, you can correct the statistics by invoking the REORG, RUNSTATS, or COPY utilities for the database objects that are described by the inaccurate statistics.

Related reference:

How utilities affect the real-time statistics

➡ RECOVER-pending status (DB2 Utilities)

➡ COPY (DB2 Utilities)

➡ REORG TABLESPACE (DB2 Utilities)

➡ REORG INDEX (DB2 Utilities)

➡ RUNSTATS (DB2 Utilities)

Part 8. Managing query access paths

The access paths that DB2 uses to process SQL statements are among most important aspects of query performance.

About this task

PSPI

The *access path* for an SQL statement specifies how DB2 accesses the data that the query specifies. It specifies the indexes and tables that are accessed, the access methods that are used, and the order in which objects are accessed.

DB2 selects the access paths for most static SQL statements when application program is bound or rebound into a package. However, access paths for some statements, such as statements that contain variable values and parameter markers, must be selected at run time. DB2 selects the access paths for dynamic SQL statements when the statements are issued.

To select efficient access paths, DB2 relies on the following elements:

- Queries that use effective predicates.
- Indexes that support efficient data access.
- Statistics that describe the data sufficiently and accurately.

Theses elements are among the foundations good query performance. It is best to verify them before you try to apply special methods to influence access path selection.

Procedure

To manage query access paths:

1. Code queries with predicates that support efficient access path selection. For more information, see Writing efficient SQL queries.
2. Ensure that accurate statistics exist to enforce the access paths for your queries. For more information, see Maintaining DB2 database statistics. Certain query optimization tools can also help you to identify statistics to support efficient access paths for your queries.
3. Create indexes that support efficient access paths for your queries. For more information, see Designing indexes for performance. Certain query optimization tools can also help you to identify indexes that support your queries.
4. For exception queries, consider applying methods to influence access path selection.

   Important: Use the following actions only after you ensure that the foundations of efficient access path selection, which are described in the previous steps, are applied.
   You might take the following actions to influence access paths, depending on the specific situation:

   - Enable queries to be reoptimized at run time, when literal values for parameter markers, host variables, and special registers might be known.

- Specify optimization parameters at the statement level.
- Specify an access path at the statement level.
- Specify an access path in a PLAN_TABLE instance.
- Add special purpose predicates to the query and apply other special methods.

What to do next

You might also apply certain methods prevent or manage access path changes.

Related concepts:

Ways to improve query performance (Introduction to DB2 for z/OS)

Investigating SQL performance by using EXPLAIN

Best Practices for Optimal Access Path Selection During Migration (DB2 for z/OS Best Practices)

Achieving Access Path Stability (DB2 for z/OS Best Practices)

Related tasks:

Investigating access path problems

Generating visual representations of access plans (IBM Data Studio)

Modeling your production system statistics in a test subsystem

Related reference:

DB2 Query Workload Tuner for z/OS

Chapter 36. Reoptimizing SQL statements at run time

You can specify whether DB2 uses literal values for host variables, parameter markers, and special registers to reoptimize SQL statements at run time.

Procedure

To manage whether DB2 re-optimizes SQL statements at run time:

1. Identify the statements that run most efficiently when DB2 follows the rules of each REOPT option.

Option	Description

REOPT(ALWAYS)	DB2 always uses literal values that are provided for parameter markers, and special registers to re-optimize the access path for any SQL statement at every execution of the statement.
	The REOPT(ALWAYS) bind option ensures the best access path for a statement. However, it can increase the cost for frequently used dynamic SQL statements.
	Use the REOPT(ALWAYS) bind option in the following circumstances:
	• The statement does not run with acceptable performance with the access path that is chosen at bind time.
	• The statement takes a relatively long time to run. For long-running statements, the performance gain from the reoptimized access can outweigh the cost of reoptimizing the access path each time that the statement runs.
	• CONCENTRATE STATEMENTS WITH LITERALS was specified when the statement was prepared but you want DB2 to consider the literal values for access path selection.
	• The dynamic statement is unlikely to result in the dynamic statement cache hits. In such cases, you might prefer to save the space in the cache for transactions that are more likely to benefit from reuse. The REOPT(ALWAYS) bind option prevents the placement of statements in the dynamic statement cache.
	• The dynamic statement cache is not enabled.
	You can issue the following statements to identify statements that are reoptimized under the REOPT(ALWAYS) bind option:
	`SELECT PLNAME,`
	`  CASE WHEN STMTNOI <> 0`
	`   THEN STMTNOI`
	`   ELSE STMTNO`
	`  END AS STMTNUM,`
	`  SEQNO, TEXT`
	`  FROM   SYSIBM.SYSSTMT`
	`  WHERE STATUS IN ('B','F','G','J')`
	`  ORDER BY PLNAME, STMTNUM, SEQNO;`
	`SELECT COLLID, NAME, VERSION,`
	`  CASE WHEN STMTNOI <> 0`
	`   THEN STMTNOI`
	`   ELSE STMTNO`
	`  END AS STMTNUM,`
	`  SEQNO, STMT`
	`  FROM SYSIBM.SYSPACKSTMT`
	`  WHERE STATUS IN ('B','F','G','J')`
	`  ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;`

Option	Description
REOPT(AUTO)	DB2 uses literal values that are provided for parameter markers, host variables, and special registers to determine at execution time whether to re-optimize access paths for cached dynamic statements. REOPT(AUTO) does not apply for static SQL statements. Use the REOPT(AUTO) bind option to achieve a better balance between the costs of reoptimization and the costs of processing a statement. You might use the REOPT(AUTO) bind option for many statements for which you might also choose either the REOPT(ALWAYS) or REOPT(NONE) bind options. Use REOPT(AUTO) especially in the following situations: • The statement is dynamic and can be cached. If dynamic statement caching is not turned on when DB2 runs a statement under the REOPT(AUTO) bind option, no reoptimization occurs. • The statement sometimes takes a relatively long time to run, depending on the values of referenced parameter markers, especially when parameter markers refer to columns that contain skewed values or that are used in range predicates. In such situations, the estimation of qualifying rows might change based on the literal values that are used at execution time. For such statements, the performance gain from a new access path that is chosen might or might not outweigh the cost of reoptimization at run time. • CONCENTRATE STATEMENTS WITH LITERALS was specified when the SQL statement was prepared but you want DB2 to consider the literal values for access path selection.

Option	Description
REOPT(ONCE)	DB2 uses literal values that are provided for parameter markers, and special registers to re-optimize cached dynamic SQL statements at run time for the first execution of the statement. The REOPT(ONCE) bind option determines the access path for an SQL statement only one time at run time and works only with dynamic SQL statements. The REOPT(ONCE) bind option allows DB2 to store the access path for dynamic SQL statements in the dynamic statement cache. REOPT(ONCE) does not apply for static SQL statements. Use the REOPT(ONCE) bind option in the following circumstances: • The SQL statement is a dynamic SQL statement. • The SQL statement does not run with acceptable performance with the access path that is chosen at bind time. • The SQL statement is relatively simple and takes a relatively short time to run. For simple statements, reoptimizing the access path each time that the statement runs can degrade performance more than using the access path from the first run for each subsequent run. • The same SQL statement is repeated many times in a loop, or is run by many threads. Because of the dynamic statement cache, the access path that DB2 chooses for the first set of input variables performs well for subsequent executions of the same SQL statement, even if the input variable values are different each time. • The application issues ad hoc non-repeating SQL statements that might use special registers, or reference views, which use special registers, such as CURRENT DATE and CURRENT TIMESTAMP. For example, such applications include DSNTEP2, DSNTIAUL, DSNTEP4, DSNTIAD, SPUFI, and QMF. • CONCENTRATE STATEMENTS WITH LITERALS was specified when the SQL statement was prepared but you want DB2 to consider the literal values for access path selection.
REOPT(NONE)	You can specify that DB2 uses the access path that was selected at run time. Statements that run with acceptable performance under the REOPT(NONE) bind option might run with even better performance under the options that might change the access path at run time. Under the REOPT(NONE) bind option DB2 does not consider literal values for access path selection when CONCENTRATE STATEMENTS WITH LITERALS is specified when statements are prepared.

2. Use one, or a combination of, the following approaches to specify the appropriate type of reoptimization for each statement:

- Create statement-level optimization parameters to specify the best reoptimization option for each statement.
- Separate the statements that run best under the different reoptimization options into different packages. You can then specify the appropriate bind option for each package.

Related concepts:

Differences between static and dynamic SQL (DB2 Application programming and SQL)

Related tasks:

Influencing access path selection

Improving dynamic SQL performance by enabling the dynamic statement cache

Using host variables efficiently

Related reference:

REOPT bind option (DB2 Commands)

Capturing reoptimized access paths

Special measures are required for capturing EXPLAIN information that accurately reflects the reoptimized access path for a SQL statement when certain REOPT options are specified.

About this task

EXPLAIN information for SQL statements that are reoptimized at run time do not always represent the access path that DB2 uses at run time. For example, in each of the following situations, the captured EXPLAIN information reflects the access path that DB2 chooses when literal values are unknown:

- For static SQL statements that use the REOPT(ALWAYS) option when EXPLAIN(YES) is specified at bind time
- For dynamic SQL statements that contain parameter markers and special registers and use the REOPT(ONCE) or REOPT(AUTO) options, when an EXPLAIN PLAN statement is issued.

Therefore, DB2 might choose a different access path at run time when the literal values are known.

Procedure

To capture EXPLAIN information that accurately reflects the access path for statements that are reoptimized at run time, use the following approaches:

- Activate performance trace class 30 (IFCID 0022 and IFCID 0063), and set a scope to limit the trace to a specific AUTHID and PLANNAME. The scope makes the collected trace records are specific to the plan and user of interest. The performance trace collects the SQL statement text and bind records that are generated each time that DB2 prepares a statement. The result shows the actual access path that DB2 generates each time that the statement is issued. You can also activate IFCID 0247 to capture the literal values that are used each time that the statements are prepared.
- For statements that are processed under the REOPT(ONCE) option, capture the EXPLAIN records from the statement in the dynamic statement cache:
  1. Locate the statement ID and token. You might issue the following statement to find these values:

```
EXPLAIN STMTCACHE ALL;
```

When you specify STMTCACHE ALL, DB2 only populates data the DSN_STATEMENT_CACHE_TABLE. No records are captured in other EXPLAIN tables.

2. Extract the EXPLAIN records from the statement cache and write them to the EXPLAIN tables. For example, you might issue one of the following statements:

```
EXPLAIN STMTCACHE STMTID statement-ID
EXPLAIN STMTCACHE STMTTOKEN statement-token
```

When you issue these statements, information is captured in the EXPLAIN tables for the specified statement.

You can also use the method for statements that use the REOPT(AUTO) option. However, the EXPLAIN records that are captured in this manner apply only to the last access path that was used. You might not be able to determine whether the retrieved access path matches the access path that was used when a problem occurred.

Related tasks:

Capturing access path information in EXPLAIN tables

Related reference:

EXPLAIN (DB2 SQL)

REOPT bind option (DB2 Commands)

Reoptimization for statements with replaced literal values

When literal values are replaced to concentrate dynamic SQL statements in the dynamic statement cache, DB2 uses the REOPT bind option to determine whether to use the literal values for access path selection.

GUPI

GUPI

PSPI

When SQL statements are prepared and the CONCENTRATE STATEMENTS WITH LITERALS clause is specified, DB2 replaces literal values in the SQL statements with ampersand symbols (&). When this replacement occurs, DB2 uses the literal values in the statement for access path selection only when the REOPT(ONCE) or REOPT(AUTO) bind options are specified. When the default option REOPT(NONE) is specified, DB2 does not consider the literal values for access path selection.

PSPI

Related concepts:

Dynamic statement cache enhancements (DB2 for z/OS What's New?)

Conditions for statement sharing

Related reference:

REOPT bind option (DB2 Commands)

PREPARE (DB2 SQL)

Related information:

Dynamic SQL literal replacement (Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS)

Chapter 37. Influencing access path selection

You can influence the access paths that DB2 uses to process SQL statements.

Before you begin

- Prepare to influence access paths.
- Create any input tables that are required.
- The following methods for influencing access path selection are best applied in exception cases, when normal access path selection results in efficient access paths. Before applying any of these methods, first take the following actions:
  - Ensure that the queries are coded with predicates for efficient access paths.
  - Collect adequate statistics to support optimal access path selection.
  - Create appropriate indexes to support efficient access paths.
  - Reoptimize the queries at run time.

About this task

When you apply any of the following methods to influence access path selection, DB2 uses information that you provide during access path selection. For static SQL statements, DB2 validates and uses the information when you rebind the package that contains the statements. For dynamic SQL statements, DB2 applies, validates, and uses the information when the statements are prepared.

You can use the BIND QUERY command to influence access path selection at the statement level. When you use these methods, DB2 applies the information for SQL statements that match the statement text you specify, in any of the following contexts:

- System-wide
- From any version of particular collection and package
- From a particular version of a collection and package

You can also insert values in a PLAN_TABLE instance to specify access paths.

Procedure

To influence the access path selection for SQL statements, use any of the following approaches:

- Specify optimization parameters for matching statements. You can use *statement-level optimization parameters* to specify the values that DB2 uses for the following options or parameters during access path selection. The parameter values apply to all matching statements in the specified context.
  - REOPT bind option
  - STARJOIN subsystem parameter
  - PARAMDEG subsystem parameter (MAX_PAR_DEGREE column)
  - CDSSRDEF subsystem parameter (DEF_CURR_DEGREE column)
  - SJTABLES subsystem parameter
- Specify access paths for matching statements. You can use *statement-level access paths* to specify that DB2 uses PLAN_TABLE rows to apply a particular access path for matching statements. Statement-level access paths are similar to

PLAN_TABLE access path hints, except that they can apply to all instances of the statement that have matching query text, at a statement or package level. DB2 stores information for matching the SQL statements and the access path information in a set of catalog tables, instead of in a PLAN_TABLE instance.

- Specify an access path in a PLAN_TABLE instance. *PLAN_TABLE access paths* try to enforce particular access paths for SQL statements that are issued by the owner of PLAN_TABLE. They use PLAN_TABLE rows to apply hints that are specified by the OPTHINT bind option, or the CURRENT OPTIMIZATION HINT special register.

Related tasks:

Creating and deploying plan hints (DB2 Query Workload Tuner for z/OS)

Related reference:

PLAN_TABLE

Default filter factors for simple predicates

BIND QUERY (DSN) (DB2 Commands)

OPTIMIZATION HINTS field (OPTHINTS subsystem parameter) (DB2 Installation and Migration)

OPTHINT bind option (DB2 Commands)

SET CURRENT OPTIMIZATION HINT (DB2 SQL)

CURRENT OPTIMIZATION HINT (DB2 SQL)

SET_PLAN_HINT stored procedure (DB2 SQL)

Preparing to influence access paths

You can specify whether a DB2 subsystem applies optimization hints and other methods for influencing the selection of access paths for SQL statements.

Procedure

PSPI

To enable management of access paths on the DB2 subsystem:

1. Set the value of the OPTHINTS subsystem parameter to 'YES'. This value is set by the OPTIMIZATION HINTS field on the performance and optimization installation panel. When you specify 'YES', DB2 enables the following actions:
   - SET CURRENT OPTIMIZATION HINT statements.
   - The OPTHINT bind option.
   - Statement-level matching for rows in the following catalog tables:
     - SYSIBM.SYSQUERY
     - SYSIBM.SYSQUERYPLAN
     - SYSIBM.SYSQUERYOPTS

   Otherwise, those actions are blocked by DB2.
2. Create a required index on instances of PLAN_TABLE that contain the access paths. The index improves the prepare performance when access path hints are used. The following example statement creates the index:

```
CREATE INDEX userid.PLAN_TABLE_HINT_IX
        ON userid.PLAN_TABLE
       ( "QUERYNO",
          "APPLNAME",
          "PROGNAME",
          "VERSION",
          "COLLID",
          "OPTHINT" )
     USING STOGROUP stogroup-name
       ERASE NO
     BUFFERPOOL BP0
     CLOSE NO;
```

The statement that creates the index is also included as part of the DSNTESC member of the SDSNSAMP library.

PSPI

Related tasks:

Influencing access path selection

Specifying access paths in a PLAN_TABLE instance

Reusing and comparing access paths at bind and rebind

Related reference:

OPTIMIZATION HINTS field (OPTHINTS subsystem parameter) (DB2 Installation and Migration)

CURRENT OPTIMIZATION HINT (DB2 SQL)

BIND and REBIND options for packages and plans (DB2 Commands)

Specifying optimization parameters at the statement level

You can customize the values of certain optimization-related subsystem parameters and bind options for all instances of particular SQL statements within particular scopes by creating statement-level optimization parameters.

Before you begin

PSPI

The following prerequisites are met:
- Prepare to manage access paths.
- You have one of the following authorities:
  - SQLADM
  - SYSOPR
  - SYSCTRL
  - SYSADM
- An instance of the DSN_USERQUERY_TABLE user table is created under your schema, or under a separate schema for input tables. For more information about using tables under a separate schema see Creating input EXPLAIN tables under a separate schemaYou can find sample CREATE statements for the tables and associated indexes in members DSNTESC and DSNTESH of the *prefix*.SDSNSAMP library.
- All object names are UPPER CASE in the query text of the SQL statements.
- The package that contains the statement was created by a BIND PACKAGE statement. Statement-level methods for influencing access paths are not

supported for statements in packages that are created by other statements, such as CREATE FUNCTION, CREATE TRIGGER, and CREATE PROCEDURE statements.

- An instance of PLAN_TABLE is created under your schema.
- An index is created on the following PLAN_TABLE columns:
  - QUERYNO
  - APPLNAME
  - PROGNAME
  - VERSION
  - COLLID
  - OPTHINT

  A sample statement that creates the index is included in member DSNTESC of the SDSNSAMP library.

About this task

Statement-level optimization parameters use matching of the statement text to apply the specified optimization parameter value to all instances of a statement within one of the following scopes:

- System-wide
- From any version of particular collection and package
- From a particular version of a collection and package

Procedure

To specify statement-level optimization parameters:

1. INSERT rows into the DSN_USERQUERY_TABLE table.
   a. Insert values in the following columns to specify the SQL statement and context for the optimization parameter:

      QUERYNO
      Specify any value that does not correlate to PLAN_TABLE rows and does not already exist in another DSN_USERQUERY_TABLE row. The QUERYNO value is used only for the primary key of DSN_USERQUERY_TABLE.

      SCHEMA
      If the SQL statement contains unqualified object names that might resolve to different default schemas, insert the schema name that identifies the unqualified database objects. If the statement contains unqualified objects names because it might apply to different schemas at different times, you must create separate hints or overrides for each possible SCHEMA value. If the statement contains only fully qualified object names, the SCHEMA value is not required. However, you can still insert a SCHEMA value to help you identify that the hint relates to a certain schema.

      QUERY_TEXT
      Insert the text of the statement whose access path you want to influence.

      The text that you provide must match the statement text that DB2 uses when binding static SQL statements and preparing dynamic SQL statements. For more information about how to enable successful text matching, see “Populating query text for statement-level matching” on page 563.

HINT_SCOPE
Insert a value to specify that context in which to match the statement.

0 System wide. DB2 uses only the text of the SQL statement and the value of the SCHEMA column, when it contains a value, to determine whether the statement matches.

1 Package-level. DB2 uses the values of the COLLECTION, PACKAGE, and VERSION columns to determine whether the statement matches.

COLLECTION
Insert the collection name of the package. This value is required only when the value of HINT_SCOPE is 1.

When the value of HINT_SCOPE is 0, the value is optional, and when a value is specified DB2 issues an error message when you bind the query if the matching value is not found in the SYSIBM.SYSPACKAGE catalog table. When HINT_SCOPE is 0, either specify both COLLECTION and PACKAGE or leave both fields blank.

For static SQL statements and dynamic SQL statements that use the DYNAMICRULES(BIND) option, you might need to specify the value of this column so that DB2 can retrieve the correct application default values from the SYSIBM.SYSPACKSTMT catalog table.

PACKAGE
Insert the name of the package. This value is required only when the value of HINT_SCOPE is 1.

When the value of HINT_SCOPE is 0, the value is optional, and when a value is specified DB2 issues an error message when you bind the query if the matching value is not found in the SYSIBM.SYSPACKAGE catalog table. When HINT_SCOPE is 0, either specify both COLLECTION and PACKAGE or leave both fields blank.

For static SQL statements and dynamic SQL statements that use the DYNAMICRULES(BIND) option, you might need to specify the value of this column so that DB2 can retrieve the correct application default values from the SYSIBM.SYSPACKSTMT catalog table.

The package-specific scope is intended primarily to support the staging, validation, and testing of statement-level hints, before they are deployed with a system-wide scope.

VERSION
Insert the version identifier of the package or '*'. A value in this column is required only when the value of HINT_SCOPE is 1. When you specify '*' for the VERSION column, DB2 does not require matching of the VERSION column for statement matching.

When the value of HINT_SCOPE is 0, this value is optional. When a value is specified DB2 issues an error message when you bind the query if the matching value is not found in the SYSIBM.SYSPACKAGE catalog table.

For static SQL statements and dynamic SQL statements that use the DYNAMICRULES(BIND) option, you might need to specify the value of this column so that DB2 can retrieve the correct application default values from the SYSIBM.SYSPACKSTMT catalog table.

   b. Insert values that define the optimization parameters. Statement-level optimization parameters are created only when the QUERYNO value that you specify in DSN_USERQUERY_TABLE does not correlate to existing PLAN_TABLE rows. You can specify settings for the following optimization parameters and options:

      - REOPT bind option
      - STARJOIN subsystem parameter
      - PARAMDEG subsystem parameter (MAX_PAR_DEGREE column)
      - CDSSRDEF subsystem parameter (DEF_CURR_DEGREE column)
      - SJTABLES subsystem parameter

      For example, you might execute the following SQL statement to insert a row into DSN_USERQUERY_TABLE that creates an optimization parameter hint:

```
INSERT INTO DSN_USERQUERY_TABLE
( QUERYNO, SCHEMA, HINT_SCOPE, QUERY_TEXT,
  USERFILTER, OTHER_OPTIONS,
  COLLECTION, PACKAGE, VERSION,
  REOPT, STARJOIN, MAX_PAR_DEGREE,
  DEF_CURR_DEGREE, SJTABLES, OTHER_PARMS )
  VALUES
  (100, 'MYSCHEMA_1', 0,
   'DECLARE C06 CURSOR FOR
     SELECT N_NAME, COUNT(*)
       FROM ORDER, CUSTOMER, NATION_NP
       WHERE C_NATIONKEY = N_NATIONKEY
         AND C_CUSTKEY = O_CUSTKEY
         AND N_REGIONKEY = :H
         AND O_ORDERDATE BETWEEN ''1998-01-01'' AND ''1998-03-31''
       GROUP BY N_NAME',
   '', '',
   '', '', '',
   'Y', '', -1,
   '', -1, '');
```

      The result is that DB2 uses the REOPT(ALWAYS) optimization parameter for instances of the specified statement.

      PSPI

2. Issue a BIND QUERY command. You must omit the LOOKUP option or specify LOOKUP(NO). DB2 takes the input from every DSN_USERQUERY_TABLE row, and from related input tables, and inserts data into the following catalog tables:

   - SYSIBM.SYSQUERY
   - SYSIBM.SYSQUERYOPTS

   The QUERYID column correlates rows in these tables.

Results

The catalog table rows for static SQL statements are validated and applied when you rebind the package that contains the statements. Catalog table rows for dynamic SQL statements are validated and enforced when the statements are prepared.

What to do next

Consider taking the following actions:

1. Validate that the appropriate catalog table rows have been created:
   a. Insert row into the DSN_USERQUERY_TABLE table that contain values in the QUERY_TEXT and SCHEMA columns.
   b. Issue the following command:
      ```
      BIND QUERY LOOKUP(YES)
      ```

      DB2 issues the following messages to indicate whether the catalog tables contain valid rows that correspond to the DSN_USERQUERY_TABLE rows.

      - A DSNT280I message for each DSN_USERQUERY_TABLE row that has matching rows in the catalog tables.
      - A DSNT281I message for each DSN_USERQUERY_TABLE row that does not have matching rows in the catalog table.
      - A single DSNT290I message if some matching rows were found in the catalog tables or a DSNT291I message if no matching rows were found.

      DB2 also updates the value of QUERYID column in the DSN_USERQUERY_TABLE table to match the value from the matching rows in the SYSIBM.SYSQUERY catalog table.
2. Delete the DSN_USERQUERY_TABLE rows to prevent the replacement of existing catalog table rows when you issue subsequent BIND QUERY commands. When you issue a BIND_QUERY command, catalog tables rows are created or replaced for every row in DSN_USERQUERY_TABLE row. Changes to data in other input tables might have unintended consequences if old rows remain in the DSN_USERQUERY_TABLE and you issue the BIND_QUERY command again.

Related tasks:

Creating and deploying plan hints (DB2 Query Workload Tuner for z/OS)

Related reference:

Tables for influencing access path selection

MAX DEGREE field (PARAMDEG subsystem parameter) (DB2 Installation and Migration)

CURRENT DEGREE field (CDSSRDEF subsystem parameter) (DB2 Installation and Migration)

STAR JOIN QUERIES field (STARJOIN subsystem parameter) (DB2 Installation and Migration)

SJTABLES in macro DSN6SPRM (DB2 Installation and Migration)

Related information:

DSNT280I (DB2 Messages)

DSNT281I (DB2 Messages)

DSNT290I (DB2 Messages)

DSNT291I (DB2 Messages)

Specifying access paths at the statement level

You can suggest that DB2 uses a certain access path for all instances of a particular SQL statement within a specified scope by creating statement-level access paths.

Before you begin

PSPI

The following prerequisites are met:

- Prepare to manage access paths.
- You have one of the following authorities:
  - SQLADM
  - SYSOPR
  - SYSCTRL
  - SYSADM
- Instances of the following user tables exist under your schema, or under a separate schema for input tables:
  - DSN_USERQUERY_TABLE
  - PLAN_TABLE

  For more information about using tables under a separate schema see “Creating input EXPLAIN tables under a separate schema” on page 565. You can find sample CREATE statements for the tables and associated indexes in members DSNTESC and DSNTESH of the *prefix*.SDSNSAMP library.
- An index is created on the following PLAN_TABLE columns:
  - QUERYNO
  - APPLNAME
  - PROGNAME
  - VERSION
  - COLLID
  - OPTHINT

  A sample statement that creates the index is included in member DSNTESC of the SDSNSAMP library.
- All object names are UPPER CASE in the query text of the SQL statements.
- The package that contains the statement was created by a BIND PACKAGE statement. Statement-level methods for influencing access paths are not supported for statements in packages that are created by other statements, such as CREATE FUNCTION, CREATE TRIGGER, and CREATE PROCEDURE statements.

About this task

Statement-level access paths use matching of the statement text to apply the specified access path to all instances of a statement within one of the following scopes:

- System-wide
- From any version of particular collection and package
- From a particular version of a collection and package

Procedure

To create statement-level access paths:

1. INSERT rows into the DSN_USERQUERY_TABLE table.
   a. Insert values in the following columns to specify the SQL statement and context in which to apply the access path:

      QUERYNO
      Insert the value that correlates to the value of the QUERYNO column of existing PLAN_TABLE rows that describe the access path that you want to enforce.

      SCHEMA
      If the SQL statement contains unqualified object names that might resolve to different default schemas, insert the schema name that identifies the unqualified database objects. If the statement contains unqualified objects names because it might apply to different schemas at different times, you must create separate hints or overrides for each possible SCHEMA value. If the statement contains only fully qualified object names, the SCHEMA value is not required. However, you can still insert a SCHEMA value to help you identify that the hint relates to a certain schema.

      QUERY_TEXT
      Insert the text of the statement whose access path you want to influence.

      The text that you provide must match the statement text that DB2 uses when binding static SQL statements and preparing dynamic SQL statements. For more information about how to enable successful text matching, see “Populating query text for statement-level matching” on page 563.

      HINT_SCOPE
      Insert a value to specify that context in which to match the statement.

      0 System wide. DB2 uses only the text of the SQL statement and the value of the SCHEMA column, when it contains a value, to determine whether the statement matches.

      1 Package-level. DB2 uses the values of the COLLECTION, PACKAGE, and VERSION columns to determine whether the statement matches.

      COLLECTION
      Insert the collection name of the package. This value is required only when the value of HINT_SCOPE is 1.

      When the value of HINT_SCOPE is 0, the value is optional, and when a value is specified DB2 issues an error message when you bind the query if the matching value is not found in the SYSIBM.SYSPACKAGE catalog table. When HINT_SCOPE is 0, either specify both COLLECTION and PACKAGE or leave both fields blank.

      For static SQL statements and dynamic SQL statements that use the DYNAMICRULES(BIND) option, you might need to specify the value of this column so that DB2 can retrieve the correct application default values from the SYSIBM.SYSPACKSTMT catalog table.

PACKAGE

Insert the name of the package. This value is required only when the value of HINT_SCOPE is 1.

When the value of HINT_SCOPE is 0, the value is optional, and when a value is specified DB2 issues an error message when you bind the query if the matching value is not found in the SYSIBM.SYSPACKAGE catalog table. When HINT_SCOPE is 0, either specify both COLLECTION and PACKAGE or leave both fields blank.

For static SQL statements and dynamic SQL statements that use the DYNAMICRULES(BIND) option, you might need to specify the value of this column so that DB2 can retrieve the correct application default values from the SYSIBM.SYSPACKSTMT catalog table.

The package-specific scope is intended primarily to support the staging, validation, and testing of statement-level hints, before they are deployed with a system-wide scope.

VERSION

Insert the version identifier of the package or '*'. A value in this column is required only when the value of HINT_SCOPE is 1. When you specify '*' for the VERSION column, DB2 does not require matching of the VERSION column for statement matching.

When the value of HINT_SCOPE is 0, this value is optional. When a value is specified DB2 issues an error message when you bind the query if the matching value is not found in the SYSIBM.SYSPACKAGE catalog table.

For static SQL statements and dynamic SQL statements that use the DYNAMICRULES(BIND) option, you might need to specify the value of this column so that DB2 can retrieve the correct application default values from the SYSIBM.SYSPACKSTMT catalog table.

For example, you might execute either of the following statements to populate DSN_USERQUERY_TABLE.

- For static SQL statements, you might retrieve values from the SYSIBM.SYSPACKSTMT catalog table and insert the values by executing a statement like the following INSERT statement:

```
INSERT INTO DSN_USERQUERY_TABLE
( QUERYNO, SCHEMA, HINT_SCOPE,
QUERY_TEXT,
USERFILTER, OTHER_OPTIONS,
  COLLECTION, PACKAGE, VERSION,
REOPT, STARJOIN,
MAX_PAR_DEGREE, DEF_CURR_DEGREE,
SJTABLES, OTHER_PARMS
)
SELECT 1111111, 'MYSCHEMA_1', 1,
STATEMENT,
'','',
COLLID, NAME, VERSION,
'', '',
-1, '', -1, ''
 FROM SYSIBM.SYSPACKSTMT
 WHERE COLLID = ' MYCOLLID_1'
 AND NAME = 'MYPACKAGE_1'
 AND VERSION = 'MYVERSION_1'
 AND STMTNO = 12;
```

When validated, the result specifies that DB2 uses the access path that is described by the PLAN_TABLE row that contains the QUERYNO value of 1111111 for instances of the specified statement that are issued under the specified package version. The 1 value for HINT_SCOPE indicates that the hint applies only to instances of the statement that are issued by the specified package.

- If the statement text and other information are not available from the SYSIBM.SYSPACKSTMT catalog table, you might issue an INSERT statement that specifies the values explicitly.

2. Populate PLAN_TABLE for the SQL statement. You can populate this table by either manually inserting one or more rows or issuing an EXPLAIN statement.
3. Issue a BIND QUERY command. You must omit the LOOKUP option or specify LOOKUP(NO). DB2 takes the input from every DSN_USERQUERY_TABLE row, and from related input tables, and inserts data into the following catalog tables:
   - SYSIBM.SYSQUERY
   - SYSIBM.SYSQUERYPLAN

   The QUERYID column correlates rows in these tables.

   PSPI

Results

The catalog table rows for static SQL statements are validated and applied when you rebind the package that contains the statements. Catalog table rows for dynamic SQL statements are validated and enforced when the statements are prepared.

If DB2 uses all of the access paths that you specified, it returns SQLCODE +394 from the PREPARE of the EXPLAIN statement and from the PREPARE of SQL statements that use the specified access paths. If any of your the specified access paths are invalid, or if any duplicates were found, DB2 issues SQLCODE +395. You can suppress SQLCODES +394 and +395 for dynamic SQL statements by setting the value of the SUPPRESS_HINT_SQLCODE_DYN subsystem parameter.

What to do next

Consider taking the following actions:

1. Validate that the appropriate catalog table rows have been created:
   a. Insert row into the DSN_USERQUERY_TABLE table that contain values in the QUERY_TEXT and SCHEMA columns.
   b. Issue the following command:

      ```
      BIND QUERY LOOKUP(YES)
      ```

      DB2 issues the following messages to indicate whether the catalog tables contain valid rows that correspond to the DSN_USERQUERY_TABLE rows.
      - A DSNT280I message for each DSN_USERQUERY_TABLE row that has matching rows in the catalog tables.
      - A DSNT281I message for each DSN_USERQUERY_TABLE row that does not have matching rows in the catalog table.
      - A single DSNT290I message if some matching rows were found in the catalog tables or a DSNT291I message if no matching rows were found.

DB2 also updates the value of QUERYID column in the DSN_USERQUERY_TABLE table to match the value from the matching rows in the SYSIBM.SYSQUERY catalog table.

2. Delete the DSN_USERQUERY_TABLE rows to prevent the replacement of existing catalog table rows when you issue subsequent BIND QUERY commands. When you issue a BIND_QUERY command, catalog tables rows are created or replaced for every row in DSN_USERQUERY_TABLE row. Changes to data in other input tables might have unintended consequences if old rows remain in the DSN_USERQUERY_TABLE and you issue the BIND_QUERY command again.

Related concepts:

Validation of specified access paths

Limitations on specified access paths

Related tasks:

Creating and deploying plan hints (DB2 Query Workload Tuner for z/OS)

Related reference:

Tables for influencing access path selection

EXPLAIN (DB2 SQL)

Related information:

DSNT280I (DB2 Messages)

DSNT281I (DB2 Messages)

DSNT290I (DB2 Messages)

DSNT291I (DB2 Messages)

+394 (DB2 Codes)

+395 (DB2 Codes)

Working with input tables for the BIND QUERY command

The BIND QUERY command uses values that you populate in certain input tables to populate catalog tables with data that influences access path selection for matching SQL statements.

Related tasks:

Preparing to influence access paths

Related reference:

Tables for influencing access path selection

BIND QUERY (DSN) (DB2 Commands)

OPTIMIZATION HINTS field (OPTHINTS subsystem parameter) (DB2 Installation and Migration)

Tables for influencing access path selection

Certain tables provide input for or contain the output when you issue BIND QUERY commands to influence access path selection for matching SQL statements.

You can use the following methods to influence access path selection for matching SQL statements:

- Specifying statement-level optimization parameters.

- Specifying statement level access paths.

Only certain tables are used, depending on the methods that you specify in the input tables.

Tables that are used for BIND QUERY input
The following tables provide input for the bind query command. The tables that you populate depend on the method that you use to influence access path selection:

DSN_USERQUERY_TABLE

PLAN_TABLE

Catalog tables that BIND QUERY commands populate
The following tables are populated by the BIND QUERY depending on the method for influencing access path selection that you specify in the input tables:

SYSIBM.SYSQUERY table (DB2 SQL)

SYSIBM.SYSQUERYOPTS table (DB2 SQL)

SYSIBM.SYSQUERYPLAN table (DB2 SQL)

Related reference:

➪ BIND QUERY (DSN) (DB2 Commands)

➪ FREE QUERY (DSN) (DB2 Commands)

➪ DB2 catalog tables (DB2 SQL)

Populating query text for statement-level matching

You can increase the likelihood that DB2 can identify matching SQL statements when you use the BIND QUERY command to influence access path selection.

About this task

DB2 modifies the statement text that is used for matching at BIND or PREPARE time. For example, white space, SQL comments, and certain clauses such as EXPLAIN, are removed from the query text. These changes enable DB2 to match the parsed SQL statements during BIND and PREPARE processing.

The following application default values must be the same at BIND QUERY time as they are when static SQL statements are bound or dynamic SQL statements are prepared:

- CCSID
- DECIMAL POINT
- STRING DELIMITER

Procedure

To enable successful matching of the statement text, use the following approaches:

- For static SQL statements, and for dynamic statements that are prepared with the DYNAMICRULES(BIND) option, specify the following columns that specify package information for the statement in DSN_USERQUERY_TABLE:
  - PACKAGE

  - COLLECTION
  - VERSION

  These values are not strictly required. However, when these values are specified, DB2 uses parsing information from the SYSIBM.SYSPACKSTMT catalog table to modify the statement text. If the values are unspecified, or the matching package is not found during the BIND QUERY processing, DB2 uses the values that are specified in the application defaults module.

  As part of the BIND QUERY process, DB2 validates that the package if specified, contains matching statement text. If the statement text does not match, DB2 issues message DSNT281I and the BIND QUERY command fails.

  When multiple versions of the package exist, and you specify * for the value of the VERSION column. DB2 uses package information from the SYSIBM.SYSPACKSTMT catalog table that has the smallest value in the VERSION column to modify the statement text. If other versions of the package use different options, it is possible that for matching to fail for statements from the other versions.

  When the package context is not specified in DSN_USERQUERY_TABLE, DB2 uses the applications default module to modify the statement text. However, the statement text is not validated against statements in a particular package.

- When you populate the QUERY_TEXT column in DSN_USERQUERY_TABLE, select the parsed query text from the following locations:
  - For static SQL statements, select the statement text from the DBRM or from the SYSIBM.SYSPACKSTMT catalog table.
  - For dynamic SQL statements, select the statement text from the dynamic statement cache. For statements that are eligible for replacement of literal values by the ampersand symbol (&), extract the statement text after DB2 replaces literal values.

  It is possible to specify the text directly in an INSERT statement (such as by copying from the source code for your application). However, that approach reduces the likelihood of successful matching of statements to the hint.

- Ensure that object names and SQL keywords in the statement text are specified by uppercase characters, especially for dynamic SQL statements.

Related tasks:

Specifying access paths at the statement level

Specifying optimization parameters at the statement level

Related reference:

DSN_USERQUERY_TABLE

DSNTIPF: Application programming defaults panel 1 (DB2 Installation and Migration)

DECIMAL POINT IS field (DECIMAL DECP value) (DB2 Installation and Migration)

STRING DELIMITER field (DELIM DECP value) (DB2 Installation and Migration)

BIND QUERY (DSN) (DB2 Commands)

SYSIBM.SYSPACKSTMT table (DB2 SQL)

DSN_STATEMENT_CACHE_TABLE

Related information:

DSNT281I (DB2 Messages)

Creating input EXPLAIN tables under a separate schema

You can simplify the task of populating EXPLAIN tables that are required for command input by creating the tables under a separate schema.

About this task

PSPI

When you issue commands such as BIND QUERY, you often must populate rows in certain related tables that contain input values. By default DB2 uses data from the EXPLAIN tables that are created under the schema of the issuer of the BIND QUERY command. Therefore, you can use the same tables as both output for EXPLAIN data and as input for the BIND QUERY command.

However, the EXPLAIN output tables are likely to contain many more rows than are needed by the BIND QUERY command. The presence of such additional rows might interfere with the hint or override that you want to create. To avoid that problem, you can specify that DB2 uses input tables under a different schema.

Procedure

To create separate tables that provide values for BIND QUERY command input:

1. Create EXPLAIN tables under your schema.
2. Create instances of the following objects under a new schema.
   - DSN_USERQUERY_TABLE table
   - PLAN_TABLE table
   - PLAN_TABLE_HINT_IX index
   - DSN_PREDICAT_TABLE table
   - DSN_PREDICAT_TABLE_IX index

   You can find sample statements for creating these objects in member DSNTESC member of the SDSNSAMP library.
3. Capture EXPLAIN information for the statements to populate the EXPLAIN tables under your schema.
4. Select only the needed rows from the EXPLAIN tables under your schema, and insert the rows into the EXPLAIN tables under the new schema. Not every table is used for every method of influencing access path selection. However, it is best to create all of the objects in the preceding list and populate only the objects that are needed.
5. Modify the rows as necessary with information that DB2 uses for access path selection. For detailed information about the rows and values that are needed, see the information about the type of hints that you want to create:
   -
   - Statement-level access paths
   - Statement-level optimization parameters
6. Issue the BIND QUERY command and specify the EXPLAININPUTSCHEMA option. For example, the following command specifies use of the input tables under the BINDQ schema:

   ```
   BIND QUERY EXPLAININPUTSCHEMA('BINDQ')
   ```

DB2 uses the values from the tables under the specified schema to populate the related catalog tables.

PSPI

Related tasks:

Influencing access path selection

Capturing access path information in EXPLAIN tables

Related reference:

Tables for influencing access path selection

BIND QUERY (DSN) (DB2 Commands)

Freeing statement-level access paths

You can remove catalog table rows that are used to influence access path selection. Doing so enables DB2 to optimize the access path for the statements, and reclaims disk space that is used to store the catalog table rows.

Before you begin

PSPI

The following prerequisites are met:

- You have one of the following authorities:
  - SQLADM
  - SYSADM
  - SYSCTRL
  - SYSOPR

Important: Catalog table rows that enforce access path reuse are also removed for specified statements. When the statements are next prepared, DB2 generates new access paths for any such statements.

Procedure

To free access paths:

Issue a FREE QUERY command, and specify one of the options that are shown in the following table:

Option	Description
To delete all existing rows	Specify the QUERYID(ALL) option.
To delete only rows for a particular query	Specify QUERYID(*queryID*), where *queryID* matches the value of the QUERYID column of the catalog tables.
To delete all rows for a group of related queries.	Specify FILTER(*value*), where *value* is the value of the FILTER column of the USERFILTER column in the SYSIBM.SYSQUERY catalog table.

PSPI

Related tasks:

Specifying access paths at the statement level

Related reference:

FREE QUERY (DSN) (DB2 Commands)

SYSIBM.SYSQUERY table (DB2 SQL)

Specifying access paths in a PLAN_TABLE instance

You try to enforce a particular access path for a SQL statement that is issued by a specific single authorization ID by creating PLAN_TABLE access paths.

Before you begin

PSPI

The following prerequisites are met.

- Prepare to manage access paths.
- An instance of the PLAN_TABLE table is created under the authorization ID that issues the SQL statement.
- An index is created on the following PLAN_TABLE columns:
  - QUERYNO
  - APPLNAME
  - PROGNAME
  - VERSION
  - COLLID
  - OPTHINT

  A sample statement that creates the index is included in member DSNTESC of the SDSNSAMP library.

About this task

When you specify a PLAN_TABLE access path, it only applies to the particular specified SQL statement, and only for instances of that statement that are issued by the authorization ID that owns the PLAN_TABLE instance that contains the specified access path.DB2 does not use the specified access path for instances of that same statement that are issued by other authorization IDs.

You can use other methods to influence access path selection to multiple instances of a statement, regardless of the authorization ID that issues the statement. For more information about specifying access paths at the statement level, see "Specifying access paths at the statement level" on page 558.

Although you might use PLAN_TABLE access paths to specify that DB2 tries to enforce the existing access path, the preferred method for preventing access path changes at rebind for static SQL statements is to specify the APREUSE bind option. Similarly, the preferred method for preventing access path change when dynamic SQL statements are prepared is to specify a plan management policy. For more information about reusing access paths, see "Reusing and comparing access paths at bind and rebind" on page 585.

Procedure

To specify access paths in a PLAN_TABLE instance:

1. Optional: Include a QUERYNO clause in your SQL statements. The following query contains an example of the QUERYNO clause:

   ```
   SELECT * FROM T1
     WHERE C1 = 10 AND
           C2 BETWEEN 10 AND 20 AND
           C3 NOT LIKE 'A%'
     QUERYNO 100;
   ```

   This step is not required for specifying access paths in PLAN_TABLE instances. However, by specifying a query number to identify each SQL statement you can eliminate ambiguity in the relationships between rows in the PLAN_TABLE and the SQL corresponding statements.

   For example, the statement number for dynamic applications is the number of the statement that prepares the statements in the application. For some applications, such as DSNTEP2, the same statement in the application prepares each dynamic statement, meaning that every dynamic statement has the same statement number.

   Similarly, when you modify an application that contains static statements, the statement numbers might change, causing rows in the PLAN_TABLE to be out of sync with the modified application. Statements that use the QUERYNO clause are not dependent on the statement numbers. You can move those statements around without affecting the relationship between rows in the PLAN_TABLE and the corresponding statements in the application.

   Such ambiguity might prevent DB2 from enforcing the specified access paths.

2. Insert a name for the specified access path in the OPTHINT column of the PLAN_TABLE rows for the SQL statement. This step enables DB2 to identify the PLAN_TABLE rows that specified the access path.

   ```
   UPDATE PLAN_TABLE
     SET OPTHINT = 'NOHYB'
     WHERE QUERYNO = 200 AND
           APPLNAME = ' ' AND
           PROGNAME = 'DSNTEP2' AND
           VERSION = ' ' AND
           COLLID = 'DSNTEP2';
   ```

3. Optional: Modify the PLAN_TABLE rows to instruct to DB2 try to enforce a different access path. You might also use PLAN_TABLE access paths only to try to enforce the same access path after a rebind or prepare. In that case, you can omit this step. However, remember that PLAN_TABLE access paths are not the recommended method for enforcing existing access paths after rebind or prepare. Use the APREUSE option at rebind, or specify a plan management policy instead.

   For example, suppose that DB2 chooses a hybrid join (`METHOD = 4`) when you know that a sort merge join (`METHOD = 2`) might perform better. You might issue the following statement.

   ```
   UPDATE PLAN_TABLE
     SET METHOD = 2
     WHERE QUERYNO = 200 AND
           APPLNAME = ' ' AND
           PROGNAME = 'DSNTEP2' AND
           VERSION = '' AND
           COLLID = 'DSNTEP2' AND
           OPTHINT = 'NOHYB' AND
           METHOD = 4;
   ```

4. Instruct DB2 to begin enforcing the specified access path:

Option	Description
For dynamic statements...	1. Issue a SET CURRENT OPTIMIZATION HINT = '*hint-name*' statement. 2. If the SET CURRENT OPTIMIZATION HINT statement is a static SQL statement, rebind the plan or package. 3. Issue an EXPLAIN statement for statements that uses the access path. DB2 adds rows to the plan table for the statement and inserts the '*hint-name*' value into the HINT_USED column. If the dynamic statement cache is enabled, DB2 tries to use the hint only when no match is found for the statement in the dynamic statement cache. Otherwise, DB2 uses the cached plan, and does not prepare the statement or consider the specified access path.
For static statements...	Rebind the plan or package that contains the statements and specify the EXPLAIN(YES) and OPTHINT('*hint-name*') options.

DB2 uses the following PLAN_TABLE columns when matching rows that specify access paths to SQL statements:

- QUERYNO
- APPLNAME
- PROGNAME
- VERSION
- COLLID
- OPTHINT

It is best to create an index on these columns for the PLAN_TABLE when you specify access paths in a PLAN_TABLE instance.

If DB2 uses all of the access paths that you specified, it returns SQLCODE +394 from the PREPARE of the EXPLAIN statement and from the PREPARE of SQL statements that use the specified access paths. If any of your the specified access paths are invalid, or if any duplicates were found, DB2 issues SQLCODE +395. You can suppress SQLCODES +394 and +395 for dynamic SQL statements by setting the value of the SUPPRESS_HINT_SQLCODE_DYN subsystem parameter.

If DB2 does not find that an access path is specified, it returns another SQLCODE. Usually, this SQLCODE is 0.DB2 also returns a message at the completion of the bind operation to identify the numbers of statements for which hints were fully applied, not applied or partially applied, and not found.

5. Select from the PLAN_TABLE to check whether DB2 used the specified access path. For example, you might issue the following statement:

```
SELECT *
  FROM PLAN_TABLE
   WHERE QUERYNO = 200
  ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The following table shows the example PLAN_TABLE data. The OPTHINT column contains the value NOHYB where the specified access path was used. You can also see that DB2 used that access path, as indicated by the NOHYB value in the HINT_USED column.

Table 115. PLAN_TABLE that shows that the NOHYB access path is used.

QUERYNO	METHOD	TNAME	OPTHINTS	HINT_USED
200	0	EMP	NOHYB	
200	2	EMPPROJACT	NOHYB	
200	3		NOHYB	
200	0	EMP		NOHYB
200	2	EMPPROJECT		NOHYB
200	3			NOHYB

PSPI

Related concepts:

Validation of specified access paths

Limitations on specified access paths

Related reference:

PLAN_TABLE

OPTIMIZATION HINTS field (OPTHINTS subsystem parameter) (DB2 Installation and Migration)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Related information:

DSNT222I (DB2 Messages)

+394 (DB2 Codes)

+395 (DB2 Codes)

Validation of specified access paths

DB2 cannot always use access paths that you specify. When a specified access path cannot be used, DB2 marks the specified access path as invalid.

PSPI If an access path that you specify has problems, DB2 invalidates it for an entire query block. In that event, DB2 determines the access path as if no access path was specified.

DB2 uses only the PLAN_TABLE columns that are shown in the following table when it validates specified access paths.

Table 116. PLAN_TABLE columns that DB2 validates

Column or columns	Accepted values or explanation
QUERYNO, APPLNAME, PROGNAME, VERSION, COLLID, OPTHINT	Must match the value of the current query number, application name, program name, version id, collection id, and the CURRENT OPTIMIZATION HINT special register respectively.

Table 116. PLAN_TABLE columns that DB2 validates (continued)

Column or columns	Accepted values or explanation
QBLOCKNO, PLANNO, and MIXOPSEQ	Must be provided to identify PLAN_TABLE rows
METHOD	Must be 0, 1, 2, 3, or 4. Any other value invalidates the specified access path.
CREATOR and TNAME	Must be specified and must name a table, materialized view, materialized nested table expression. Blank if METHOD is 3. If a table is named that does not exist or is not involved in the query, then the specified access paths are invalid.
MATCHCOLS	This value is used only when ACCESSTYPE is IN, N, or HN. The value must be greater than or equal to 0.
TABNO	This value is required only when the following columns do not uniquely identify a particular table reference: • CREATOR • TNAME • CORRELATION_NAME • QBLOCKNO This situation might occur when the same table is referenced within multiple views with the same CORRELATION_NAME. This field is ignored when it is not needed.
ACCESSTYPE	The access method for the table. For the list of accepted values, see PLAN_TABLE. Each row that contains one of the following values must be preceded by a row that contains ACCESSTYPE='M', or the specified access path is invalidated: • DI • DU • MH • MI • MU • MX Any row that contains ACCESSTYPE='A' invalidates the specified access path.
ACCESSCREATOR and ACCESSNAME	Ignored if ACCESSTYPE is R or M. If ACCESSTYPE contains any of the following values, then these fields must identify an index on the specified table. • H • HN • I • I1 • IN • N • NR If the index does not exist, or if the index is defined on a different table, then the specified access paths are invalid. Also, if the specified index cannot be used, the specified access paths are invalid.

Table 116. PLAN_TABLE columns that DB2 validates (continued)

Column or columns	Accepted values or explanation
SORTN_JOIN and SORTC_JOIN	Must be Y, N or blank. Any other value invalidates the specified access path. This value determines if DB2 should sort the new (SORTN_JOIN) or composite (SORTC_JOIN) table. This value is ignored if the specified join method, join sequence, access type and access name dictate whether a sort of the new or composite tables is required.
SORTC_ORDERBY and SORTC_GROUPBY	For a list of accepted values, see PLAN_TABLE. Unexpected values are ignored.
PREFETCH	This value determines the type of prefetch that DB2 uses. For a list of accepted values, see PLAN_TABLE. This value is ignored if the specified access type and access name dictates the type of prefetch required.
COLUMN_FN_EVAL	For a list of accepted values, see PLAN_TABLE. Unexpected values are ignored.
PAGE_RANGE	Must be Y, N or blank. Any other value invalidates the specified access path.
JOIN_TYPE	For a list of accepted values, see PLAN_TABLE. Unexpected values are ignored.
PARALLELISM_MODE	This value is used only if it is possible to run the query in parallel; that is, the SET CURRENT DEGREE special register contains ANY, or the plan or package was bound with DEGREE(ANY). If parallelism is possible, this column must contain one of the following values: • C • I • X • null All of the restrictions involving parallelism still apply when using access path hints. If the specified mode cannot be performed, the specified access path are either invalidated or the mode is modified. A possible result is that the query runs without parallelism. A null value indicates no parallelism. If the plan table contains multiple specifications for parallelism, DB2 uses only the first one. DB2 does not compare multiple specified access paths to check consistency.

Table 116. PLAN_TABLE columns that DB2 validates (continued)

Column or columns	Accepted values or explanation
ACCESS_DEGREE or JOIN_DEGREE	If PARALLELISM_MODE is specified, use this field to specify the degree of parallelism. If you specify a degree of parallelism, this must a number greater than zero, and DB2 might adjust the parallel degree from what you set here.A null value indicates no parallelism. If you want DB2 to determine the degree, do not enter a value in this field. If you specify a value for ACCESS_DEGREE or JOIN_DEGREE, you must also specify a corresponding ACCESS_PGROUP_ID and JOIN_PGROUP_ID. If DB2 uses a hint for parallelism, other than for IN-list parallelism or a hint for degree=0, DB2 uses the hinted degree of parallelism unconditionally, regardless of the value of the PARAMDEG subsystem parameter. Consequently, be careful to ensure that hints specify reasonable degrees of parallelism.
SORTN_PGROUP_ID and SORTC_PGROUP_ID	Must be a positive number or null. A null value indicates no parallel sort for the corresponding table.
WHEN_OPTIMIZE	Must be R, B, or blank. Any other value invalidates the specified access path. When a statement in a plan that is bound with REOPT(ALWAYS) qualifies for reoptimization at run time, and you have provided optimization hints for that statement, the value of WHEN_OPTIMIZE determines whether DB2 reoptimizes the statement at run time. If the value of WHEN_OPTIMIZE is blank or B, DB2 uses only the access path that is provided by the optimization hints at bind time. If the value of WHEN_OPTIMIZE is R, DB2 uses the specified access paths at bind time to determine the access path. At run time, DB2 searches the PLAN_TABLE for hints again, and if specified access paths for the statement are still in the PLAN_TABLE and are still valid, DB2 optimizes the access path again.
QBLOCK_TYPE	A value must be specified. A blank in the QBLOCK_TYPE column invalidates the specified access path.
PRIMARY_ACCESSTYPE	Must be D, T, or blank. Any other value invalidates the specified access paths.
MERGC	Must be Y, or N.

PSPI

Related tasks:

Specifying access paths at the statement level

Specifying access paths in a PLAN_TABLE instance

Related reference:

PLAN_TABLE

Limitations on specified access paths

DB2 cannot always apply the information that you provide to influence access path selection.

PSPI

In certain situations, DB2 cannot apply specified access paths, and hints might be used differently from release to release, or not at all in certain releases. For example:

- Query transformations, such as subquery transformation to join or materialization or merge of a view or table expression, cannot be forced or undone.

  A query that is not transformed in one release of DB2 might be transformed in a later release of DB2. If you use a hint in one release for a query that is not transformed, but the query is transformed in a later release, DB2 cannot use the hint in the later release.
- DB2 might apply a specified access path differently in different releases, and an equivalent access path might not look the same in both releases. For example, before DB2 9 DB2 work files that represent non-correlated subqueries were not shown in the PLAN_TABLE. An access path that is based on the same hint that was used in earlier version might contain a row for such work file in newer releases.
- DB2 ignores any PLAN_TABLE row that contains METHOD=3.
- When access paths are specified, DB2 ignores any PLAN_TABLE row that contains the following values for the QBLOCK_TYPE column:
  - INSERT
  - UNION
  - UNIONA
  - INTERS
  - INTERA
  - EXCEPT
  - EXCEPTA
- If the PLAN_TABLE contains multiple rows that specify for parallelism, DB2 uses only the first one. It does not compare multiple rows to check for consistency.
- If parallelism is specified, DB2 uses the hinted degree of parallelism unconditionally, regardless of the value of the PARAMDEG subsystem parameter. IN-list parallelism and hints for DEGREE=0 are exceptions to this rule. Use care to specify only reasonable degrees of parallelism.

PSPI

Related concepts:

Query transformations

Related tasks:

Programming for parallel processing

Related reference:

PLAN_TABLE

MAX DEGREE field (PARAMDEG subsystem parameter) (DB2 Installation and Migration)

Interactions of methods for influencing access paths

Statement level-methods for influencing access path selection and PLAN_TABLE optimization hints can be used in combination for the same statements. However, certain restrictions apply.

Catalog table rows that influence access paths for matching statements can coexist with PLAN_TABLE optimization hints that apply to the same statements. DB2 also creates and uses rows in the same catalog tables when you specify access path reuse at bind or rebind.

Different methods of influencing access paths can coexist. However, DB2 applies only one method. When more than one coexisting method might apply to the same statement, only the first that applies is used, in the following order of precedence:

1. PLAN_TABLE access path hints
2. Statement-level access paths or parameters for a specific version, collection, and package.
3. Statement-level access paths or parameters for a specific collection and package.
4. Statement-level access paths or parameters that have a system-wide scope.
5. Statement-level access paths that are created internally by DB2 for access path reuse

However, you can use only one statement-level method to influence the access path for the same SQL statement in the same application context. This rule applies to the following methods of influencing access paths:

- Specifying statement-level optimization parameters.
- Specifying statement level access paths.

When you issue a BIND QUERY command, DB2 replaces existing SYSIBM.SYSQUERY catalog table rows that apply to the same statement in the same application context.

For example, assume that SYSIBM.SYSQUERY catalog tables already contains a row that specifies a statement where *SQL-text* statement text, and S1 schema:

```
STMTTEXT  SCHEMA  PACKAGE
SQL-text  S1
```

Assume that you then issue BIND QUERY commands in sequence when the DSN_USERQUERY table contains the specified rows:

1. A row that specifies only the *SQL-text*. DB2 replaces the existing row. The schema-specific catalog table rows are replaced by global rows for matching statements. The SYSIBM.SYSQUERY table still contains only one row.

   ```
   STMTTEXT  SCHEMA  PACKAGE
   SQL-text
   ```

2. A row that specifies *SQL-text* and the S2 schema. The global catalog table rows are replaced by the new schema-specific rows. The SYSIBM.SYSQUERY table now contains only one row:

   ```
   STMTTEXT  SCHEMA  PACKAGE
   SQL-text  S2
   ```

3. A row that specifies *SQL-text* and the S1 schema. The new row is inserted and does not replace the existing row. The SYSIBM.SYSQUERY table now contains two rows:

```
STMTTEXT  SCHEMA  PACKAGE
SQL-text  S2
SQL-text  S1
```

4. A row that specifies *SQL-text*, the S1 schema, and the P1 package. This row replaces the existing row for the same statement and schema that was created in step 3. The SYSIBM.SYSQUERY table now contains two rows:

```
STMTTEXT  SCHEMA  PACKAGE
SQL-text  S2
SQL-text  S1      P1
```

5. A row that specifies *SQL-text*, the S1 schema, and the P2 package. DB2 inserts a new row. The SYSIBM.SYSQUERY tables now contains three rows:

```
STMTTEXT  SCHEMA  PACKAGE
SQL-text  S2
SQL-text  S1      P1
SQL-text  S1      P2
```

6. A row that specifies only *SQL-text* and the S1 schema. The new row replaces both rows that were created in steps 4 and 5. The SYSIBM.SYSQUERY table now contains only two rows:

```
STMTTEXT  SCHEMA  PACKAGE
SQL-text  S2
SQL-text  S1
```

Related tasks:

Specifying access paths in a PLAN_TABLE instance

Related reference:

DSN_USERQUERY_TABLE

PLAN_TABLE

BIND QUERY (DSN) (DB2 Commands)

SYSIBM.SYSQUERY table (DB2 SQL)

OPTIMIZATION HINTS field (OPTHINTS subsystem parameter) (DB2 Installation and Migration)

SET CURRENT OPTIMIZATION HINT (DB2 SQL)

CURRENT OPTIMIZATION HINT (DB2 SQL)

Modifying catalog statistics to influence access path selection

If you have the proper authority, you can influence access path selection by using an SQL statements to change statistics values in the catalog. However, doing so is not generally recommended, except as a last resort.

Before you begin

PSPI

Important: Use care when issuing SQL statements or using tools to update statistics values in catalog tables. If such updates introduce invalid data, unpredictable results can occur, including abends for RUNSTATS and other utilities. If such problems occur, you can run the RUNSTATS utility and collect statistics at the table space level to resolve the problems, in most cases.

Important: The access path selection techniques that are described here might cause significant performance degradation if they are not carefully implemented and monitored. Also, access path selection methods might change in a later release of DB2, causing your changes to degrade performance.

Consequently, the following recommendations apply if you make any such changes:

- Save the original catalog statistics or SQL statements before you consider making any changes to control the choice of access path.
- Before and after you make any changes, take performance measurements.
- Be prepared to back out any changes that have degraded performance.
- When you migrate to a new release, evaluate the performance again.
- Plan to keep track of the changes you make and of the plans or packages that have access path changes because of the changed statistics.
- Record when the statistics were modified by setting the value of the STATIME column to the current TIMESTAMP value for every statistics record that you modify in the catalog tables.
- Consider correlations among catalog tables before updating statistics values.

About this task

You might modify catalog statistics to influence how DB2 selects access paths. However, the access path selection "tricks" that are described here cause significant performance degradation if they are not carefully implemented and monitored. Although modifying catalog statistics might improve the access path for one or a few SQL statements, other statements in the workload might by affected by the same changes. Also, the updates to the catalog must be repeated whenever the RUNSTATS utility resets the catalog values.

Procedure

To modify statistics values, use any of the following approaches:

- Issue UPDATE statements to change the values in the catalog. Numeric, date time, values in the catalog tables use internal hexadecimal formats. You must also allow for null indicators in keys that allow null values.

  Related information:

  Dates, times, and timestamps for edit and validation routines (DB2 Administration Guide)

  DB2 codes for numeric data in edit and validation routines (DB2 Administration Guide)

  Null values for edit procedures, field procedures, and validation routines (DB2 Administration Guide)

  Similarly, the padding characteristics of values in columns such as HIGH2KEY and LOW2KEY, that contain data values must match the padding characteristics of the corresponding data columns. You can query the STATS_FORMAT column of SYSIBM.SYSCOLUMNS catalog table to determine whether variable-length columns contain padded values.
- Update small COLCARDF values for partitioned data. On partitioned indexes, the RUNSTATS INDEX utility calculates the number of distinct column values

and saves it in the SYSCOLSTATS.COLCARD, by partition. When statistics by partition are used to form an aggregate COLCARDF value, the aggregate value might not be exact because some column values might occur in more than one partition. DB2 cannot detect that overlap without scanning all parts of the index. The overlap never skews COLCARD by more than the number of partitions, which is usually not a problem for large values.

The same problem and solution also applies to the FIRSTKEYCARDF columns of the SYSIBM.SYSINDEXES and SYSIBM.SYSINDEXSTATS catalog tables.

- If you update the COLCARDF value for a column, also update HIGH2KEY and LOW2KEY for the values.
- You can insert, update, or delete distribution information for any column in the following catalog tables that contain distributions statistics:
  - SYSIBM.SYSCOLDIST
  - SYSIBM.SYSCOLDISTSTATS
  - SYSIBM.SYSKEYTGTDISTSTATS
  - SYSIBM.SYSKEYTGTDIST

  Related information:

  Filter factors for all distributions

  SYSIBM.SYSCOLDIST table (DB2 SQL)

  SYSIBM.SYSCOLDISTSTATS table (DB2 SQL)

  SYSIBM.SYSKEYTGTDIST table (DB2 SQL)

  SYSIBM.SYSKEYTGTDISTSTATS table (DB2 SQL)
- If you use dynamic statement caching, invalidate statements in the cache that involve the table spaces or indexes whose statistics you have modified. To invalidate statements in the dynamic statement cache without updating catalog statistics or generating reports, you can run the RUNSTATS utility with the REPORT NO and UPDATE NONE options on the table space or the index that the query is dependent on.

Example

For example, assume that the following SQL statement has a problem with data correlation:

```
SELECT * FROM PART_HISTORY   --  SELECT ALL PARTS
WHERE PART_TYPE = 'BB'    P1 --  THAT ARE 'BB' TYPES
  AND W_FROM = 3          P2 --  THAT WERE MADE IN CENTER 3
  AND W_NOW = 3           P3 --  AND ARE STILL IN CENTER 3
```

DB2 does not know that 50% of the parts that were made in Center 3 are still in Center 3. The problem can be circumvented by making a predicate non-indexable. However, suppose that hundreds of users are writing queries similar to that query. Having all users change their queries would be impossible. In this type of situation, the best solution might be to change the catalog statistics.

One catalog table that you can update is the SYSIBM.SYSCOLDIST catalog table, which gives information about a column or set of columns in a table. Assume that because columns W_NOW and W_FROM are correlated, and that only 100 distinct values exist for the combination of the two columns, rather than 2500 (50 for W_FROM * 50 for W_NOW). Insert a row like this to indicate the new cardinality:

```
INSERT INTO SYSIBM.SYSCOLDIST
       (FREQUENCY, FREQUENCYF, IBMREQD,
        TBOWNER, TBNAME, NAME, COLVALUE,
        TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
 VALUES(0, -1, 'N',
        'USRT001','PART_HISTORY','W_FROM',' ',
        'C',100,X'00040003',2);
```

You can also use the RUNSTATS utility to put this information in SYSCOLDIST.

You tell DB2 about the frequency of a certain combination of column values by updating SYSIBM.SYSCOLDIST. For example, you can indicate that 1% of the rows in PART_HISTORY contain the values 3 for W_FROM and 3 for W_NOW by inserting this row into SYSCOLDIST:

```
INSERT INTO SYSIBM.SYSCOLDIST
       (FREQUENCY, FREQUENCYF, STATSTIME, IBMREQD,
        TBOWNER, TBNAME, NAME, COLVALUE,
        TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
 VALUES(0, .0100, '2006-12-01-12.00.00.000000','N',
        'USRT001','PART_HISTORY','W_FROM',X'00800000030080000003',
        'F',-1,X'00040003',2);
```

PSPI

Related tasks:

Improving filter factors by collecting cardinality and frequency statistics

Setting default statistics for created temporary tables

Related reference:

Statistics used for access path selection

RUNSTATS (DB2 Utilities)

UPDATE (DB2 SQL)

Correlations in the catalog

Important relationships exist among certain columns in DB2 catalog tables. Consider these relationships if you choose to modify the catalog statistics to achieve a more favorable access path.

PSPI

Correlations exist among columns in the following catalog tables:
- Columns within the SYSIBM.SYSCOLUMNS catalog table
- Columns in the SYSIBM.SYSCOLUMNS and SYSIBM.SYSINDEXES catalog tables
- Columns in the tables SYSIBM.SYSCOLUMNS and SYSIBM.SYSCOLDIST catalog tables
- Columns in the tables SYSIBM.SYSCOLUMNS, SYSIBM.SYSCOLDIST, and SYSIBM.SYSINDEXES catalog tables
- Columns with table space statistics and columns for partition-level statistics.

If you plan to modify the values of statistics in the catalog tables, consider the following correlations:

COLCARDF and FIRSTKEYCARDF

For a column that is the first column of an index, those two values are equal. If the index has only that one column, the two values are also equal to the value of FULLKEYCARDF.

COLCARDF, LOW2KEY, and HIGH2KEY

If the COLCARDF value is not '-1' or'-2', DB2 assumes that statistics exist for the column. In particular, it uses the values of LOW2KEY and HIGH2KEY in calculating filter factors. If COLDARDF = 1 or if COLCARDF = 2, DB2 uses HIGH2KEY and LOW2KEY as domain statistics, and generates frequencies on HIGH2KEY and LOW2KEY.

CARDF in SYSCOLDIST

CARDF is related to COLCARDF in SYSIBM.SYSCOLUMNS and to FIRSTKEYCARDF and FULLKEYCARDF in SYSIBM.SYSINDEXES. CARDF must be the minimum of the following values:

- A value between FIRSTKEYCARDF and FULLKEYCARDF if the index contains the same set of columns
- A value between MAX(COLCARDF of each column in the column group) and the product of multiplying together the COLCARDF of each column in the column group

Example: Assume the following set of statistics:

```
CARDF = 1000
NUMCOLUMNS = 3
COLGROUPCOLNO = 2,3,5

INDEX1 on columns 2,3,5,7,8
FIRSTKEYCARDF = 100                CARDF must be between 100
FULLKEYCARDF = 10000                                and 10000

column 2 COLCARDF = 100
column 3 COLCARDF =  50
column 5 COLCARDF =  10
```

The range between FIRSTKEYCARDF and FULLKEYCARDF is 100 and 10,000. The maximum of the COLCARDF values is 50,000. Thus, the allowable range is 100 - 10,000.

CARDF in SYSTABLES

CARDF must be equal to or larger than any of the other cardinalities, SUCH AS COLCARDF, FIRSTKEYCARDF, FULLKEYCARDF, and CARDF in SYSIBM.SYSCOLDIST.

FREQUENCYF and COLCARDF or CARDF

The number of frequencies collected must be less than or equal to COLCARDF for the column or CARDF for the column group.

FREQUENCYF

The sum of frequencies collected for a column or column group must be less than or equal to 1.

PSPI

Related reference:

SYSIBM.SYSCOLUMNS table (DB2 SQL)

SYSIBM.SYSINDEXES table (DB2 SQL)

SYSIBM.SYSCOLDIST table (DB2 SQL)

Chapter 38. Managing and preventing access path change

You can prevent unwanted access path changes for critical applications when you rebind applications that contain static SQL statements and when DB2 prepares dynamic SQL statements.

About this task

PSPI

Query optimization depends on many factors, and even minor changes in the database environment might cause significant changes to access paths. DB2 uses the statistics that are stored in the DB2 catalog to determine the most efficient access paths during the bind process. When you reorganize your data, collect statistics, and rebind your packages or plans, DB2 can choose the most efficient access paths for your queries.

Because DB2 must often rely on incomplete information, such as statistics, suboptimal access paths are possible. Reoptimization sometimes yields access paths that cause performance regressions, including unnecessary I/O operations and excessive processor consumption, and even application outages.

DB2 also considers the following system and subsystem attributes during access path selection:

- Central processor model
- The number of central processors (for determining the appropriate degree of parallelism.)
- Buffer pool size, and other statistics
- RID pool size

These factors can change access paths for a statement from one system to another, even if all the catalog statistics are identical. So, you must account for these factors when you model your production systems on test systems, and when you model new applications. Mixed central processor models in a data sharing group might also affect access path selection.

PSPI

Related concepts:

Interpreting data access by using EXPLAIN

Related tasks:

Investigating access path problems

Maintaining data organization and statistics

Tuning database buffer pools

Managing RID pool size

Modeling a production environment on a test subsystem

Modeling your production system statistics in a test subsystem

Related reference:

Statistics used for access path selection

Managing access path change at migration from DB2 9

You can specify that DB2 tries to reuse existing access paths, and detect when access paths have changed, when you rebind your applications at migration.

About this task

GUPI

How you manage your access paths at migration depends on the amount of change that your applications can tolerate. The most cautious approach is to try to enforce the previous access paths. However, this approach might also prevent your applications from using new access paths that might indeed provide improved performance.

When you want to reuse access paths for existing applications, use of the REBIND command is preferred over use of the BIND command with the REPLACE option. The REBIND command provides more reliable plan management techniques that cannot be used with the BIND command.

Procedure

To manage access path changes at migration, use one of the following approaches:

- Retain the existing access paths from the previous version, or reject the rebind:
  1. Specify the EXTENDED plan management policy to retain existing access paths at rebind. DB2 saves information about the existing access paths.
  2. Issue a FREE PACKAGE command to free inactive copies of the package. Otherwise, the existing original copy remains and the current active copy might be lost at a subsequent rebind.
  3. Issue REBIND commands to rebind the applications, and specify the APREUSE (ERROR) and APCOMPARE (ERROR) options. DB2 tries to use hints to enforce the previous access paths. If any hints cannot be applied, or access path that results from the hint does not match the previous access path, processing ends for the package and DB2 issues messages.
- Consider new access paths, but detect access path changes:
  1. Specify the EXTENDED plan management policy to retain existing access paths at rebind. DB2 populates the access path repository with information about the existing access paths.
  2. Issue a FREE PACKAGE command to free inactive copies of the package. Otherwise, the existing original copy remains and the current active copy might be lost at a subsequent rebind.
  3. Issue REBIND commands to rebind the applications and specify the APCOMPARE(WARN) option. DB2 does not try to reuse existing access paths and issues messages when access paths change.
- Detect whether to expect access paths change before rebinding applications:
  1. Issue REBIND commands, but specify the EXPLAIN(ONLY) and APCOMPARE(WARN) bind options. The rebind action is not completed. Instead, DB2 issues messages for access paths that cannot be reused after a rebind.You can use the PLAN_TABLE data to identify the changed access paths.

Important: When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied. To see the new access paths that DB2 uses for statements when no reuse is applied, you can specify APREUSE(NONE) and EXPLAIN(ONLY) to populate the PLAN_TABLE with values that describe the new access paths that result from clean optimizations.

2. Issue REBIND commands to rebind only applications that can successfully reuse all access paths, or require only acceptable access path changes.

GUPI

What to do next

It is best to issue FREE PACKAGE commands to free inactive package copies when the migration is complete. By doing so you can reclaim the disk space that was used by the inactive copies.

Related concepts:

Plan management polices

Best Practices for Optimal Access Path Selection During Migration (DB2 for z/OS Best Practices)

Related tasks:

Freeing saved access paths for static SQL statements

Rebinding a package (DB2 Application programming and SQL)

Reusing and comparing access paths at bind and rebind

Analyzing access path changes at bind or rebind

Migrating DB2 to DB2 10 (DB2 Installation and Migration)

Related reference:

REBIND PACKAGE (DSN) (DB2 Commands)

REBIND TRIGGER PACKAGE (DSN) (DB2 Commands)

BIND and REBIND options for packages and plans (DB2 Commands)

Managing access paths at migration from DB2 Version 8

You can specify that DB2 retains the access paths from the previous version in case an access path change results in performance regression.

About this task

You can save information about existing access paths in the access path repository and actively switch back to the previous access path when a regression occurs. You cannot specify that DB2 tries to actively reuse existing access paths when you migrate from DB2 Version 8.

Procedure

To manage access path changes when migrating from DB2 Version 8.

1. Issue a REBIND PACKAGE command to rebind the applications and specify the PLANMGMT(EXTENDED) bind option to retain information about existing

access paths at rebind. If you use a BIND command with the ACTION(REPLACE) option as an alternative, the plan management policy is not invoked. Therefore, plan management and access path reuse techniques cannot be applied. DB2 populates the access path repository with information about the existing access paths.

2. When access path changes result in performance regression, rebind the application and specify the SWITCH option to revert to the prior access path.
3. Optional: When the rebind is complete with acceptable access paths, issue a FREE PACKAGE command and specify the INACTIVE option to remove unneeded access path information from the access path repository.

Related concepts:

Introduction to migration from Version 8 (DB2 Installation and Migration)

Plan management polices

Changes that might affect your migration from Version 8 (DB2 Installation and Migration)

Related tasks:

Rebinding a package (DB2 Application programming and SQL)

Migrating DB2 to DB2 10 (DB2 Installation and Migration)

Related reference:

REBIND PACKAGE (DSN) (DB2 Commands)

BIND and REBIND options for packages and plans (DB2 Commands)

FREE PACKAGE (DSN) (DB2 Commands)

Managing access path changes for periodic maintenance

You can detect and evaluate access path changes when you rebind after organizing your data and collecting statistics, and when you apply routine maintenance. You can also specify that DB2 tries to reuse existing access paths when you rebind. You can also save information about existing access paths when you rebind, and revert to previous access paths when performance regressions occur.

About this task

Reorganizing your data and capturing accurate statistics are important activities for maintaining good performance. Generally new access paths offer improvements because the organization and statistics for the data have changed.

Certain other maintenance activities (such as applying PTFs) might also require that you rebind your applications, before you can take advantages of the fixes that are delivered. Again, it is possible that new access paths after the rebind provide better performance that the existing access paths.

However, unwanted access path changes sometimes result from these changes. You can investigate the access path changes before rebinding, or specify that DB2 tries to reuse the existing access paths.

Procedure

To prevent access path regression when you apply DB2 maintenance, use one of the following approaches:

- Retain the existing access paths and rebuild the run time structures:
  1. Issue a FREE PACKAGE command to free inactive copies of the package. Otherwise, the existing original copy remains and the current active copy might be lost at a subsequent rebind.
  2. Issue a REBIND PACKAGE command, and specify the following options:
     - PLANMGMT(EXTENDED) to specify that DB2 saves information about the existing access paths.
     - APREUSE(ERROR) to specify that DB2 actively tries to reuse access paths or APCOMPARE(ERROR) to specify that DB2 accepts only unchanged access paths.

     Optionally, you can the specify EXPLAIN(ONLY) option to investigate access path changes without completing the specified bind operation.
- Consider new access paths, but detect access path changes:
  1. Issue a FREE PACKAGE command to free inactive copies of the package. Otherwise, the existing original copy remains and the current active copy might be lost at a subsequent rebind.
  2. Issue a REBIND PACKAGE command, and specify the following options:
     - PLANMGMT(EXTENDED) to specify that DB2 saves information about the existing access paths. The BIND command with the ACTION(REPLACE) option does not support this capability.
     - APCOMPARE(WARN) to specify that DB2 identifies access paths that change after the rebind operation.
  3. When undesirable access path changes occur, rebind again and use the SWITCH option to revert to the previous access paths.

What to do next

It is best to issue FREE PACKAGE commands to free inactive package copies when the application of maintenance is complete. By doing so you can reclaim the disk space that was used by the inactive copies.

Related concepts:

Plan management polices

Interpreting data access by using EXPLAIN

Related tasks:

Reusing and comparing access paths at bind and rebind

Analyzing access path changes at bind or rebind

Maintaining data organization and statistics

Switching to previous access paths

Rebinding a package (DB2 Application programming and SQL)

Related reference:

REBIND PACKAGE (DSN) (DB2 Commands)

BIND and REBIND options for packages and plans (DB2 Commands)

EXPLAIN (DB2 SQL)

Reusing and comparing access paths at bind and rebind

You can specify that DB2 tries to reuse previous access paths for SQL statements whenever possible, and compares the new and previous access paths.

Before you begin

PSPI

The following prerequisites have been met:

- The package that contains the SQL statements was created in DB2 9 or later.

About this task

In many cases, you must rebind packages before your applications can benefit from performance improvements from the latest version. Similarly, you must bind packages again after modifying your applications to take advantage of functions from a new version of DB2. However, a common complaint is that the necessary bind or rebind operations often result in access path changes, and these changes sometimes result in unacceptable performance regressions.

However, you can reduce the risk that is associated with rebinding packages by using a reuse or comparison strategy when you rebind your packages. With *access path reuse*, DB2 automatically specifies internal statement-level access paths to try to enforce previous access paths. When a BIND command is used to bind a package that contains both statements that existed before and new statements, DB2 tries to reuse only for the pre-existing statements.

Important: Access path reuse is not guaranteed to succeed in all cases. For example an access path that relies on objects (such as indexes) that no longer exist cannot be reused. Version incompatibilities might also prevent access paths from being reused. Some access paths cannot be reused because of ambiguity in the underlying access path information. For example: an access path specifies to the type of join to use and the number of matching columns, but the names of the matching columns are not available.

With *access path comparison* DB2 verifies that a new access path matches the previous access path.

DB2 uses the comparison when you specify reuse to validate that the access path that results from successful reuse actually matches the previous access path. However, you can also use access comparison separately when you want to know how access paths have changed without preventing those changes.

Procedure

To prevent and assess changes to access paths changes at bind or rebind, you can use one or a combination of the following approaches:

- Specify the APREUSE(ERROR) bind option. Under this option, which is the most cautious approach for preventing access path changes, you accept rebinds only when all access paths for a package can be reused. Under the APREUSE(ERROR) bind option, the rebind operation fails for a package after all of the statements are processed whenever any access paths cannot be reused. DB2 begins processing the next package, if additional packages are specified. Because comparison is also implied by reuse, processing for a package might end because a comparison fails. When all processing is complete, DB2 issues a DSNT286I message to report the numbers of access paths that could and could not be reused.
- Specify the APCOMPARE(WARN) option. This approach means that you do not want DB2 to try to reuse old access paths, but that you do want to identify

access path changes. Some access paths might change after the rebind operation, but you are willing to accept all changes.

Consequently, the following recommendations apply to this approach:

– Use a REBIND PACKAGE command and specify a plan management policy that enables you to switch to previous access paths if unacceptable changes occur.

– Examine the quantity and types of changes to look for potential problems. You can specify the EXPLAIN(YES) option capture information about the selected access paths.

– Maintain a list of packages with changed access paths to aid in problem isolation when a regression occurs.

However, when the bind operation is complete, DB2 issues a DSNT285I message that indicates how many access paths are changed after the rebind operation.

- Specify the EXPLAIN(YES) option in combination with APREUSE(ERROR) and APCOMPARE(ERROR) or (WARN). You can use EXPLAIN output to analyze the access paths in more detail.

  Important: When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied. To see the new access paths that DB2 uses for statements when no reuse is applied, you can specify APREUSE(NONE) and EXPLAIN(ONLY) to populate the PLAN_TABLE with values that describe the new access paths that result from clean optimizations. DB2 inserts information about reuse and comparison failures into the PLAN_TABLE.REMARKS column.

- Specify the EXPLAIN(ONLY) option in combination with the APREUSE(ERROR) or an APCOMPARE option. This approach enables you to learn whether access path changes occur before you actually bind or rebind the packages.

  Important: When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied. To see the new access paths that DB2 uses for statements when no reuse is applied, you can specify APREUSE(NONE) and EXPLAIN(ONLY) to populate the PLAN_TABLE with values that describe the new access paths that result from clean optimizations. DB2 processes all statements in the packages, and inserts information about any failures in the PLAN_TABLE.REMARKS column. However, the bind or rebind operation is not completed. Therefore, under EXPLAIN(ONLY), both APCOMPARE(ERROR) and APCOMPARE(WARN) have the same effect, because the new packages are not created.

What to do next

Examine DSNT285I and DSNT286I messages. For more detailed analysis, query the following PLAN_TABLE columns to identify the access path changes that resulted from the bind or rebind:

- REMARKS
- HINT_USED
- BIND_EXPLAIN_ONLY

Important: When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied. To see the new access paths that DB2 uses for statements when no reuse is applied, you can specify APREUSE(NONE) and EXPLAIN(ONLY) to populate the PLAN_TABLE with values that describe the new access paths that result from clean optimizations.

When you have completed your analysis, you must decide the approach to take for any packages that failed because APREUSE(ERROR) was specified. For example, you might take any of the following approaches:

- Rebind the remaining packages and specify no reuse. This approach enables your packages to take advantage of new functions, but it exposes every statement in the packages to possible access path changes.
- Do not rebind the packages.

PSPI

Related concepts:

Achieving Access Path Stability (DB2 for z/OS Best Practices)

Related tasks:

Rebinding a package (DB2 Application programming and SQL)

Specifying access paths at the statement level

Reverting to saved access paths for static SQL statements

Related reference:

BIND and REBIND options for packages and plans (DB2 Commands)

REBIND PACKAGE (DSN) (DB2 Commands)

REBIND TRIGGER PACKAGE (DSN) (DB2 Commands)

Related information:

DSNT285I (DB2 Messages)

DSNT286I (DB2 Messages)

How DB2 identifies packages for reuse under BIND PACKAGE commands

When you specify access path reuse in a BIND PACKAGE command, DB2 tries to identify and reuse access paths from existing packages.

PSPI

When a BIND PACKAGE command specifies the APREUSE(ERROR) bind option, DB2 tries to locate the access path information from a package that has a matching identity based on the following package information criteria:

- Location
- Collection identifier
- Name
- Version

If no such package exists, DB2 tries to locate the most recently created other version of the package that has the matching location, collection ID, and name. When a prior version of a matching package is reused, DB2 issues a DSNT294I message.

When a prior version exists, the set of static SQL statements in a previous version might not be identical to the set of statements in the new package, including white space in the statement text. In that situation, the APREUSE option only applies to statements that have identical statement text in both packages. Newly added statements and statements with text changes never reuse previous access paths.

DB2 issues a DSNT292I warning message when it cannot locate any previous package, and ignores the APREUSE option for that package.

PSPI

Related reference:

REBIND PACKAGE (DSN) (DB2 Commands)

BIND and REBIND options for packages and plans (DB2 Commands)

Related information:

DSNT292I (DB2 Messages)

DSNT294I (DB2 Messages)

Analyzing access path changes at bind or rebind

When you rebind packages, you can use access path comparison to learn which access paths have changed, and to verify whether access path reuse was successful.

Before you begin

PSPI

The following prerequisites have been met:
- The package that contains the SQL statements was created in DB2 9 or later.

About this task

With *access path comparison* DB2 verifies that a new access path matches the previous access path. The comparison examines a new access path and compares it to the existing access path. When DB2 detects mismatches between new and previous access paths, it writes information about the mismatches to the PLAN_TABLE.REMARKS column.

When the comparison is complete, DB2 issues a DSNT285I message that indicates how many access paths are changed and unchanged after the rebind operation.

You can specify either error or warning rules for comparison. Under warning rules, the bind operations complete regardless of whether reuse or comparison fails for statements in a package. Under error rules, the bind operations ends for a package after all statements are processed if any reuse or comparison error is encountered. You can specify only error rules for reuse.

DB2 also uses access path comparison to validate the resulting access paths when you specify access path reuse.

Procedure

To assess the result of access path changes after bind or rebind:

1. Specify one of the following bind options to activate access path comparison:
   - APCOMPARE(WARN) to get new access paths and find out when changes occur. This option might result in unacceptable access path changes. Therefore, the recommendation is to also use a plan management policy that enables you to switch to previous access paths.
   - APCOMPARE(ERROR) to accept no access path changes, without trying to reuse.
   - APREUSE(ERROR) to try to enforce the reuse of access paths, accepting no reuse failures or access path changes
2. Specify one of the following bind options so that DB2 creates EXPLAIN output for the bind or rebind operations:
   - EXPLAIN(ONLY) specifies that the bind and rebind operations are not completed. However, DB2 processes the access paths and populates PLAN_TABLE with access path comparison result information. Because the bind or rebind operation are never completed under this option, APCOMPARE(ERROR) and APCOMPARE(WARN) have the same meaning when EXPLAIN(ONLY) is specified.
3. Issue the command to bind or rebind the packages. The REBIND PACKAGE command is preferred when access path changes are accepted because it enables the use of plan management policies.
4. Examine the DSNT285I message to learn the numbers of access paths that are changed and unchanged by the bind or rebind operations.

   ```
   DSNT285I  -csect-name REBIND FOR PACKAGE = package-name,
             USE OF APCOMPARE RESULTS IN:
             75 STATEMENTS WHERE COMPARISON IS SUCCESSFUL
             5 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL
             2 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.
   ```

   If you specified access path reuse, you can also examine the DSNT286I message which provides statistics about the success of reuse.

   ```
   DSNT286I  -csect-name REBIND FOR PACKAGE = package-name,
             USE OF APREUSE RESULTS IN:
             73 STATEMENTS WHERE APREUSE IS SUCCESSFUL
             7 STATEMENTS WHERE APREUSE IS EITHER NOT SUCCESSFUL
               OR PARTIALLY SUCCESSFUL
             2 STATEMENTS WHERE APREUSE COULD NOT BE PERFORMED
             0 STATEMENTS WHERE APREUSE WAS SUPPRESSED BY OTHER HINTS.
   ```

   Although DB2 also uses comparison when you specify access path reuse, the comparison and reuse are separate operations. Reuse fails when DB2 cannot apply a hint to enforce the previous access path. Comparison fails when the new access path does not match the previous access path. Notice, as shown in the preceding example, that even when reuse fails DB2 might sometimes select a new access path that matches the previous access path.
5. Query the PLAN_TABLE to find rows that DB2 populated as a result of the comparison or reuse.

   When reuse succeeds, DB2 inserts the value 'APREUSE' into the PLAN_TABLE.HINT_USED column. When reuse or comparison fails, DB2 populates the PLAN_TABLE.REMARKS column with information about the failures:
   - When reuse fails, DB2 inserts a reason code that corresponds to the reason codes values from SQLCODE +395. When reuse fails, the resulting

PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied.

- When comparison fails, DB2 inserts information about the mismatch. When corresponding rows for the new and previous access path do not match, DB2 identifies a column that contains the values that do not match:
  - For rows that exist in both the old and new access paths, DB2 identifies the mismatch in the REMARKS column for the corresponding row for the new access path.
  - For rows that exist only in the new access path, the unmatched row is indicated in the REMARKS column of the new nonmatching row.
  - For rows that exist only for the old access path, the unmatched row is identified in the REMARKS column of a different row for the new access path.

Important: When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied. To see the new access paths that DB2 uses for statements when no reuse is applied, you can specify APREUSE(NONE) and EXPLAIN(ONLY) to populate the PLAN_TABLE with values that describe the new access paths that result from clean optimizations.

What to do next

You might take any of the following actions based on the results of the comparison:

- When reuse fails, rebind packages that can tolerate changes without any reuse. In this case you can use APCOMPARE(WARN) instead. You can use the message and PLAN_TABLE output to create a list of packages and statements that had access path changes. The list might be helpful for isolating problems when performance regressions occur.
- When reuse fails for packages that cannot tolerate access path changes, do not rebind the packages. This option is the most cautious approach, but it might prevent your applications from taking advantage of performance improvements.

Related reference:

PLAN_TABLE

BIND and REBIND options for packages and plans (DB2 Commands)

BIND PACKAGE (DSN) (DB2 Commands)

REBIND PACKAGE (DSN) (DB2 Commands)

REBIND TRIGGER PACKAGE (DSN) (DB2 Commands)

Related information:

+395 (DB2 Codes)

DSNT285I (DB2 Messages)

Rebinding packages when access path reuse fails

When you rebind packages and encounter reuse failures, you must next decide what to do about the remaining packages.

Before you begin

Capture and analyze EXPLAIN information for packages that cannot be rebound with the APREUSE(ERROR) option.

About this task

Certain limitations apply to access path reuse, and not all access paths can be reused successfully. So, successful reuse is not guaranteed for all statements. However, access path reuse under the APREUSE(ERROR) bind option is an all-or-nothing proposition at the package level.

Reuse sometimes fails when a hint can be applied only partially. The result might be an access path that matches neither the existing access path nor the new access path that DB2 chooses, when reuse is not applied. For that reason, reuse succeeds for all matching statements access paths, or the rebind option fails for the entire package.

Some of the approaches described here include detailed analysis of individual packages and access paths for specific statements. You might find that the costs of such activities are prohibitive.

Procedure

To handle packages that encounter reuse failures, use one or a combination of the following approaches:

- Rebind the packages and specify no reuse. This approach enables your applications to take advantage of new functions at migration. It also enables DB2 to choose new access paths based on the latest statistics. However, because all access paths in the package are subject to change, care must be taken to evaluate whether the applications can tolerate the potential changes.
  1. Reissue the command to rebind the packages and specify the APREUSE(NONE), APCOMPARE(WARN), and EXPLAIN(ONLY) bind options. The rebind operation is not completed, but DB2 populates the PLAN_TABLE with information about the new access paths.

     Important: When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied. To see the new access paths that DB2 uses for statements when no reuse is applied, you can specify APREUSE(NONE) and EXPLAIN(ONLY) to populate the PLAN_TABLE with values that describe the new access paths that result from clean optimizations.
  2. Analyze the PLAN_TABLE records for the new access paths, and determine whether the new access paths are acceptable.
  3. If the new access paths are acceptable, reissue the command to rebind the packages and specify APREUSE(NONE). DB2 generates new access paths for all statements in the packages.
- Rebind the package and accept new access paths for the reuse failures, but create hints to enforce existing access paths for some statements in the package. This advanced approach might enable you to enforce the existing access paths for some statements when not all access paths can be reused. However, it requires a

significant amount of effort for detailed analysis and implementation. Care must be taken to identify access paths that cannot be reused, and to exclude those statements when you create the hints:

  1. Reissue the command to rebind the package and specify the APREUSE(ERROR) and EXPLAIN(ONLY) options. The rebind operation is not completed, but DB2 populates the PLAN_TABLE with information about the reuse failures.
  2. Analyze the PLAN_TABLE data to identify the statements that encountered reuse failures, and determine whether access path changes can be tolerated for those statements. When reuse fails, the resulting PLAN_TABLE information reflects the access paths that result from the failed application of the underlying hints. Therefore, the values do not represent the previous access paths or the new access path that DB2 selects when reuse is not applied.
  3. Reissue the command to rebind the packages and specify APCOMPARE(WARN) and EXPLAIN(ONLY) options. The rebind operation is not completed, but DB2 populates the PLAN_TABLE with information about the new access paths.
  4. Analyze the PLAN_TABLE data to evaluate the new access paths for the statements that encountered reuse failures. You might even find that DB2 chooses the same access path for a statement through normal optimization, even though reuse could not be enforced.
  5. Use the PLAN_TABLE information to create hints for the access paths that can be reused in the package.

     Important: Do not create hints to try to enforce access paths for statements that encounter reuse failures. Such hints are likely to be only partially applied, if at all. They are likely to result in access paths that do not match either the existing access path or the new access path that DB2 chooses.

- Do not rebind the packages that encounter reuse failures. This approach is the most conservative approach in terms of preventing access paths changes, and it requires the least effort for detailed analysis of your packages. However, taking this approach at migration is likely to prevent your applications from taking advantage of enhancements that are delivered by the new version.

Related reference:

BIND and REBIND options for packages and plans (DB2 Commands)

PLAN_TABLE

Switching to previous access paths

You can save information about the previous access paths when you use a REBIND command to rebind your packages. When a new access path results in a performance regression, you can switch back to the previous access path.

About this task

When you specify a plan management policy, DB2 saves access path information for SQL statements when you issue a REBIND PACKAGE or REBIND TRIGGER PACKAGE command to rebind. When an access path change results in a performance regression, you can specify the SWITCH option and rebind the package again to revert to the previous access paths.

The use of plan management polices is recommend whenever you rebind packages with the APCOMPARE(WARN) option. This approach enables you to revert the package to use the previous access paths when you detect access path changes that you do not want.

Related tasks:

Reusing and comparing access paths at bind and rebind

Related reference:

REBIND PACKAGE (DSN) (DB2 Commands)

REBIND TRIGGER PACKAGE (DSN) (DB2 Commands)

BIND and REBIND options for packages and plans (DB2 Commands)

Plan management polices

You can use plan management policies enable you to save historical information about the access paths for SQL statements.

Plan management policy options

PSPI

You can specify the following *plan management policies*:

OFF No copies are saved.

BASIC
DB2 saves the active copy and one older copy, which is known as the previous copy.DB2 replaces the previous copy path with the former active copy and saves the newest copy the active copy.

EXTENDED
DB2 saves the active copy, and two older copies, which are known as the previous and original copies. replaces DB2 the previous copy with the former active copy and saves the newest copy as the active copy. DB2 saves the original copy at the first rebind that EXTENDED is specified and retains the original copy unchanged at subsequent rebinds. EXTENDED is the default value of the PLANMGMT subsystem parameter.

Methods for specifying plan management policies

You can use several different methods to specify a plan management policy and plan management scope. Each of the methods can be used in combination with the other methods:

PLANMGMT subsystem parameter
This value applies to all statements member-wide unless they are overridden by a value specified by one of the other methods. When you change the value of the subsystem parameter, the changes applies only to packages that are bound or rebound after the parameters is changed.

PLANMGMT rebind option
For static or dynamic SQL statements, you can rebind packages and specify the PLANMGMT bind option. If you do not specify the bind options, the value that is specified for the subsystem parameter of the same name is used when you bind or rebind a package.

PSPI

Related tasks:

Saving access path information for static SQL statements

Related reference:

PLANMGMT bind option (DB2 Commands)

Package copies

When a package is bound, DB2 stores relevant package information for static SQL statements as records in several catalog tables and in the directory.

PSPI

DB2 records the following types of information about static SQL statements when you bind a package:
- Metadata, in the SYSIBM.SYSPACKAGES catalog table
- Query text, in the SYSIBM.SYSPACKSTMT catalog table
- Dependencies, in the SYSIBM.SYSPACKDEP catalog table
- Authorizations
- Access paths
- Compiled run time structures, in the DSNDB01.SPT01 directory table space

When you rebind a package, DB2 deletes many of these records and replaces them with new records. However, you can specify a plan management policy to specify that DB2 retains such records in *package copies* when you rebind packages. The default plan management policy is EXTENDED, which means that DB2 saves as many as three copies for each package. When you specify EXTENDED for the plan management policy, DB2 retains *active*, *previous*, and *original* copies of the package.

Each copy might contain different metadata and compiled run time structures. However, the following attributes are common to all copies in a corresponding set of active, previous, and original package copies:

- Location
- Collection
- Package name
- Version
- Consistency token

You can use package copies with the following types of applications:

- Regular packages
- Stored procedures
- Trigger packages

Invalid package copies

Packages copies can become invalid when any object that the package depends on is altered or dropped. For example, a package can become invalid when an index or table is dropped, or when a table column is altered. The SYSIBM.SYSPACKDEP catalog table records package dependencies. Depending on the change, different copies of the packages can be affected differently. For a dropped table, all of the copies of the package would be invalidated. Whereas, in the case of a dropped index, only certain copies that depend on the dropped index might be invalidated.

Packages copies can also become invalid if they were last bound or rebound in a DB2 release that is too old. In DB2 10, package copies that were bound in releases prior to Version 6 Release 1 are invalid.

When DB2 finds that the active copy for a package is invalid, it uses an automatic bind to replace the current copy. The automatic bind is not affected by invalid status on any previous or original copies, and the automatic bind replaces only the active copy.

PSPI

Related concepts:

Plan management polices

Automatic rebinds (DB2 Application programming and SQL)

Related tasks:

Reverting to saved access paths for static SQL statements

Related reference:

SWITCH bind option (DB2 Commands)

SYSIBM.SYSPACKAGE table (DB2 SQL)

SYSIBM.SYSPACKSTMT table (DB2 SQL)

SYSIBM.SYSPACKDEP table (DB2 SQL)

Saving access path information for static SQL statements

You can use package copies to automatically save pertinent catalog table and directory records for static SQL statements when you bind a package or rebind an existing package.

About this task

PSPI

When a performance regression occurs after you rebind a package, you can use the saved historical information to switch back to the older copy of the package and regain the old access paths. You can also use the historical information in conjunction with the APREUSE bind option to specify that DB2 tries to reuse the existing active access paths for static SQL statements when you rebind packages.

Procedure

To save access path information for static SQL statements:

1. Specify the PLANMGMT bind option when you issue one of the following commands:
   - REBIND PACKAGE
   - REBIND TRIGGER PACKAGE

   The values of the PLANMGMT option have the following meanings:

   OFF No copies are saved.

BASIC
DB2 saves the active copy and one older copy, which is known as the previous copy.DB2 replaces the previous copy path with the former active copy and saves the newest copy the active copy.

EXTENDED
DB2 saves the active copy, and two older copies, which are known as the previous and original copies. replaces DB2 the previous copy with the former active copy and saves the newest copy as the active copy. DB2 saves the original copy at the first rebind that EXTENDED is specified and retains the original copy unchanged at subsequent rebinds. EXTENDED is the default value of the PLANMGMT subsystem parameter.

When you rebind a package with the PLANMGMT (BASIC) or (EXTENDED) options, the following options must remain the same for package copies to remain usable:

- OWNER
- QUALIFIER
- ENABLE
- DISABLE
- PATH
- PATHDEFAULT
- IMMEDWRITE

2. Optional: Specify the APRETAINDUP(NO) option reduce the amount of disk space that is consumed by package copies. When you specify the APRETAINDUP option, DB2 saves only previous and original copies of the access path information as package copies when they differ from the active copy.

PSPI

Related concepts:

Package copies

Related reference:

BIND and REBIND options for packages and plans (DB2 Commands)

Reverting to saved access paths for static SQL statements

In the event that package that contains static SQL statements suffers performance regression after a rebind, you can fall back to a copy of the better performing access paths for the package.

Before you begin

PSPI

The following prerequisites have been met:

- You previously specified a plan management policy to specify that DB2 saves access path information.

Procedure

To revert a package to use previously saved access paths, use one of the following approaches:

- Specify the SWITCH option when you issue one of the following commands:

- REBIND PACKAGE
- REBIND TRIGGER PACKAGE

You can specify one of the following options:

Option	Description
SWITCH(PREVIOUS)	DB2 toggles the active and previous packages: • The existing active copy takes the place of the previous copy • The existing previous copy takes the place of the active copy. • Any existing original copy remains unchanged.
SWITCH(ORIGINAL)	DB2 replaces the active copy with the original copy: • The existing active copy replaces the previous copy. • The existing previous copy is discarded. • The existing original copy remains unchanged.

You can use wild cards (*) in the syntax to restore the previous or original packages for multiple packages. When you specify the SWITCH option, package copies that are stored as rows of data in the following objects are modified:

- DSNDB01.SPT01 table space
- SYSIBM.SYSPACKDEP catalog table
- SYSIBM.SYSPACKCOPY catalog table

If no previous or original package copies exist, DB2 issues a DSNT217I error message for each package the does not have copies and processes the remainder of the packages normally.

PSPI

- If the previous or original package copy is invalid, or it was bound in a release prior to Version 6 Release 1, take the following actions:
  1. Rebind the package and specify the SWITCH bind option that corresponds to the invalid package.
  2. Rebind the package again and specify the APREUSE(WARN) or APREUSE(ERROR) bind option.

Related concepts:

Package copies

Related reference:

SWITCH bind option (DB2 Commands)

REBIND PACKAGE (DSN) (DB2 Commands)

SYSIBM.SYSPACKCOPY table (DB2 SQL)

SYSIBM.SYSPACKDEP table (DB2 SQL)

Related information:

DSNT217I (DB2 Messages)

Freeing saved access paths for static SQL statements

You can remove saved copies of access path information for static SQL statements to free up the disk space that is used to save them.

Before you begin

PSPI

The following prerequisites have been met:

- You previously specified a plan management policy to specify that DB2 saves access path information.

About this task

When you save access path copies for static SQL statements, some disk space is used to save the copies. The amount of space that is used depends on the plan management policy that you choose. When you specify an extended plan management policy, the amount of space used might be triple the amount used when you specify only that plan management is on, if you do not specify the APRETAINDUP option when you bind the packages. Consequently, you might want to remove some or unneeded historical copies of the access path information to free disk space.

Procedure

To remove saved copies of access path information for static SQL statements:

Issue a FREE PACKAGE command.

- Specify the PLANMGMTSCOPE value that indicates the copies that you want to free. You can specify whether you want to free all package copies or inactive package copies only.

  For example, you might issue the following command to free the disk space that is used by inactive package copies:

  ```
  FREE PACKAGE (collection-name.package-name) PLANMGMTSCOPE (INACTIVE)
  ```

The PLANMGMTSCOPE option cannot be used for remote processing.

PSPI

Related concepts:

Package copies

Related tasks:

Saving access path information for static SQL statements

Related reference:

FREE PACKAGE (DSN) (DB2 Commands)

APRETAINDUP bind option (DB2 Commands)

Part 9. Monitoring and analyzing DB2 performance data

You can monitor DB2 to detect performance problems and undertake detailed analysis of performance data.

Related tasks:

Monitoring the use of accelerators for DB2 for z/OS queries

Related information:

➪ DB2 for z/OS System and Application Monitoring and Tuning

Chapter 39. Planning for performance monitoring

When you plan to monitor DB2 performance, you should consider how to monitor performance continuously, how and when to perform periodic monitoring, how you will monitor exceptions, and the costs associated with monitoring.

Your plan for monitoring DB2 performance should include:

- A master schedule of monitoring. Large batch jobs or utility runs can cause activity peaks. Coordinate monitoring with other operations so that it need not conflict with unusual peaks, unless that is what you want to monitor.
- The kinds of analysis to be performed and the tools to be used. Document the data that is extracted from the monitoring output. These reports can be produced using Tivoli Decision Support for z/OS, IBM Tivoli Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS XE, other reporting tools, manual reduction, or a program of your own that extracts information from standard reports.
- A list of people who should review the results. The results of monitoring and the conclusions based on them should be available to the user support group and to system performance specialists.
- A strategy for tuning DB2 describes how often changes are permitted and standards for testing their effects. Include the tuning strategy in regular system management procedures.

  Tuning recommendations might include generic database and application design changes. You should update development standards and guidelines to reflect your experience and to avoid repeating mistakes.

Cost factors of performance monitoring

You should consider the following cost factors when planning for performance monitoring and tuning.

- Trace overhead for global, accounting, statistics, audit, and performance traces
- Trace data reduction and reporting times
- Time spent on report analysis and tuning action

Related tasks:

Minimizing the processing cost of DB2 traces

Related reference:

Facilities and tools for DB2 performance monitoring

Continuous performance monitoring

Continuous monitoring watches system throughput, resource usage (processor, I/Os, and storage), changes to the system, and significant exceptions that might affect system performance.

Procedure

- Try to continually run classes 1, 3, 4, and 6 of the DB2 statistics trace and classes 1 and 3 of the DB2 accounting trace. For data sharing environments, run class 5 too.

- In the data that you collect, look for statistics or counts that differ from past records.
- Pay special attention to peak periods of activity, both of any new application and of the system as a whole
- Run accounting class 2 as well as class 1 to separate DB2 times from application times.

  Running with CICS without the open transaction environment (OTE), entails less need to run with accounting class 2. Application and non-DB2 processing take place under the CICS main TCB. Because SQL activity takes place under the SQL TCB, the class 1 and class 2 times are generally close. The CICS attachment work is spread across class 1, class 2, and time spent processing outside of DB2. Class 1 time thus reports on the SQL TCB time and some of the CICS attachment. If you are concerned about class 2 overhead and you run a CICS workload only, you can generally run without turning on accounting class 2.
- Run accounting class 3 trace for monitoring suspension time in DB2.
- Run accounting class 7 and 8 if you want to monitor package information. Class 10 provides more detailed package information but also has additional CPU and SMF volume overhead.
- Statistics and accounting information can be very helpful for application and database designers. Consider putting this information into a performance warehouse so that the data can be analyzed more easily by all the personnel who need the information.

  IBM Tivoli Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS DB2 Performance Expert on z/OS includes a performance warehouse that allows you to define, schedule, and run processes that help in monitoring performance trends and tuning.

  The data in the performance warehouse can be accessed by any member of the DB2 family or by any product that supports Distributed Relational Database Architecture™ (DRDA).

Planning for periodic monitoring

Periodic monitoring serves to check system throughput, used resources (processor, I/Os, and storage), changes to the system, and significant exceptions that might affect system performance during peak periods when constraints and response-time problems are more pronounced.

About this task

A typical periodic monitoring interval of about ten minutes provides information on the workload achieved, resources used, and significant changes to the system. In effect, you are taking “snapshots” at peak loads and under normal conditions.

The current peak is also a good indicator of the future average. You might have to monitor more frequently at first to confirm that expected peaks correspond with actual ones. Do not base conclusions on one or two monitoring periods, but on data from several days representing different periods.

You might notice that subsystem response is becoming increasingly sluggish, or that more applications fail from lack of resources (such as from locking contention or concurrency limits). You also might notice an increase in the processor time DB2 is using, even though subsystem responses seem normal. In any case, if the subsystem continues to perform acceptably and you are not having any problems, DB2 might not need additional tuning.

Procedure

To monitor peak periods:

Gather information from the different parts of your system, including:
- DB2 for z/OS
- z/OS
- The transaction manager (IMS, CICS, or WebSphere)
- DB2 Connect™
- IBM Data Server Driver for JDBC and SQLJ
- The network
- Distributed application platforms (such as Windows, UNIX, or Linux)

To compare the different results from each source, monitor each for the same period of time. Because the monitoring tools require resources, you need to consider the overhead for using these tools as well.

Related tasks:

Minimizing the processing cost of DB2 traces

Detailed performance monitoring

You can add detailed monitoring to periodic monitoring when you discover or suspect a problem. You can also use detailed monitoring to investigate areas that are not covered by your periodic monitoring. To minimize the cost of the detailed monitoring, limit the information to the specific application and data as much as possible.

Procedure

- If you have a performance problem, first verify that it is not caused by faulty design of an application or database.
- If you believe that the problem is caused by the choice of system parameters, I/O device assignments, or other factors, begin monitoring DB2 to collect data about its internal activity.
- If you have access path problems, use the query tuning and visual explain features of IBM Data Studio, or the DB2 EXPLAIN facility to locate and tune the problems.

Related tasks:

Planning for and designing DB2 applications (DB2 Application programming and SQL)

Related reference:

Facilities and tools for DB2 performance monitoring

Related information:

Designing a database (DB2 Administration Guide)

Monitoring for performance exceptions

By maintaining a performance database and analyzing performance trends for your DB2 subsystems, you can identify potential problem conditions before they become actual problems. By detecting such problems early, you can either prevent problems or react to and troubleshoot problems more quickly when they do occur.

About this task

You can monitor for specific exceptional values or events, such as high response times or deadlocks. Exception monitoring is most appropriate for response-time and concurrency problems.

Procedure

Use any combination of the following approaches for exception performance monitoring in your DB2 subsystems:

- Collect statistics trace classes 1, 3, 4, 5 (for data sharing), 7 (for distributed location statistics), and 8 (for dataset statistics).
- Set the value of the STATIME subsystem parameter to 1. The STATIME subsystem parameter specifies the time interval, in minutes, between statistics collections. Starting in DB2 10, DB2 always uses a 1-minute interval for certain statistics values, including the following IFCIDs: 0002, 0202, 0217, 0225, and 0230.
- Copy SMF 100 records and retain them in a separate file. These records represent a relatively small volume of the total SMF data volume.
- Collect accounting trace classes 1, 2, 3, 7, and 8.
- Use the ACCUMACC subsystem parameter to create rollup accounting records for DDF and RRS workloads. Be aware, however, that using the rollup accounting records provides a trade-off. The rollup records reduce the volume of the accounting data. However, they also remove granularity from the data, which means that information about outlying transactions that perform poorly is likely to be lost in the rollup data.

  Another way to reduce the volume of accounting data is to set the value of the SMFCOMP subsystem parameter to YES. This value enables DB2 SMF trace data compression. You can use it instead of ACCUMACC to reduce the volume of trace data without removing the granularity of the data. You can also use SMFCOMP=YES and ACCUMACC=YES together to get even more SMF trace volume reduction.
- Monitor and review the DB2 metrics proactively and regularly, and develop an automated process to store the performance data into DB2 tables. The goal is to track evolving trends for key performance indicators at the system level from the time that DB2 starts to the time that it stops. This information becomes the baseline for further analysis and enables you to establish thresholds for out-of-normal conditions and alerts when these conditions occur.
- Enable near-term history collection in your DB2 online monitor. By doing so, you can review DB2 statistics and accounting records for the past several hours of DB2 processing. You can use this information to detect adverse trends. You can keep a log and history of the alerts that are generated, and analyze the trends.
- Use automation to issue the DISPLAY commands at regular intervals. Save the output from these commands so that you can analyze what happened on the system before the problem occurred. You might also use automation to specify thresholds for the output of the commands to detect exception conditions. The objective is to be able to detect and correct problems quickly, thereby avoiding long-running recovery actions whenever possible. For example, you might use automation to issue the following DISPLAY commands at 15-minute intervals:

  ```
  -DISPLAY THREAD(*) SCOPE(GROUP) TYPE(INDOUBT)
  -DISPLAY DATABASE(*) SPACENAM(*) RESTRICT LIMIT(*)
  -DISPLAY UTIL(*)
  -DISPLAY THREAD(*) TYPE(SYSTEM)
  ```

```
-DISPLAY THREAD(*) SERVICE(STORAGE)
-DISPLAY BPOOL(*) DETAIL(INTERVAL)
-DISPLAY GBPOOL(*) MDETAIL(INTERVAL)
-DISPLAY GROUP
-DISPLAY ARCHIVE
-DISPLAY DDF
-DISPLAY LOCATION(*) DETAIL
F irlmproc,STATUS, ALLD
D XCF, STR
D GRS,C
```

- Use the dynamic statement cache to monitor for exceptions for distributed applications, such as .NET, ODBC, and JDBC applications. For example, you might use the following process to capture the information:
  1. Create the following DB2-supplied user tables:
     - DSN_STATEMENT_CACHE_TABLE
     - DSN_STATEMNT_TABLE
     - DSN_FUNCTION_TABLE
     - PLAN_TABLE

     You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.
  2. Issue the following command to start collecting performance statistics for the dynamic statement cache:
     ```
     -START TRACE(P) CLASS(30) IFCID(316, 317, 318)
     ```
  3. Issue the following statement to extract the statistics information from the global statement cache to the DSN_STATEMENT_CACHE_TABLE.
     ```
     EXPLAIN STMTCACHE ALL;
     ```
  4. Stop the performance trace.
  5. Issue the following statement to generate individual EXPLAIN statements for each SQL statement in the cache.
     ```
     SELECT 'EXPLAIN STMTCACHE STMTID '||STRIP(CHAR(STMT_ID))||' ;'
     FROM USERID.DSN_STATEMENT_CACHE_TABLE;
     ```
  6. Import the contents of the four tables into a spreadsheet for analysis.

Related concepts:

Accounting trace

Near-term history information (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Working with the Performance Warehouse (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related tasks:

Recording SMF trace data

Enabling Near-Term History (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Using Statistics and Accounting reports to identify exceptions (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Capturing performance information for dynamic SQL statements

Related reference:

SMF STATISTICS field (SMFSTAT subsystem parameter) (DB2 Installation and Migration)

STATISTICS TIME field (STATIME subsystem parameter) (DB2 Installation and Migration)

DDF/RRSAF ACCUM field (ACCUMACC subsystem parameter) (DB2 Installation and Migration)

COMPRESS SMF RECS field (SMFCOMP subsystem parameter) (DB2 Installation and Migration)

-START TRACE (DB2) (DB2 Commands)

Related information:

Exception processing (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Chapter 40. Facilities and tools for DB2 performance monitoring

You can use various tools and facilities to monitor DB2 activity and performance.

Table 117. Monitoring tools in a DB2 for z/OS environment

Data source	Data Writer (if applicable)	Reporting Tool (if applicable)	Reduction and History
CICS monitoring facility	SMF or CICS journal	Tivoli Decision Support for z/OS	Tivoli Decision Support for z/OS
IMS monitor		IMS Performance Analyzer for z/OS IMS Monitor Report Print utility (DFSUTR20)	
IMS log records	IMS log	IMS Performance Analyzer for z/OS	
DB2 trace	SMF or GTF	Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS	
DB2 RUNSTATS utility			
DB2 STOSPACE utility			
EXPLAIN Dynamic Statement Cache		IBM Data Studio IBM Data Server Manager DB2 Query Workload Tuner for z/OS Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS	
DB2 DISPLAY commands		Message output	
DB2 catalog tables		DSNACCOX stored procedure	
RMF monitor	SMF	RMF reports	

DB2 for z/OS facilities for performance monitoring

DB2 trace

Provides DB2 accounting, statistics, auditing, and performance trace information.

Related information:

DB2 trace

DB2 trace output

RUNSTATS utility

Can report space use and access path statistics in the DB2 catalog.

Related information:

RUNSTATS (DB2 Utilities)

Maintaining DB2 database statistics

Automating statistics maintenance

STOSPACE utility

Provides information about the actual space allocated for storage groups, table spaces, table space partitions, index spaces, and index space partitions.

Related information:

STOSPACE (DB2 Utilities)

DISPLAY commands

Provide information about the status of things such as threads, databases, buffer pools, traces, allied subsystems, applications, and the allocation of tape units for the archive read process.

Related information:

-DISPLAY ACCEL (DB2) (DB2 Commands)

-DISPLAY ARCHIVE (DB2) (DB2 Commands)

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

-DISPLAY DATABASE (DB2) (DB2 Commands)

-DISPLAY DDF (DB2) (DB2 Commands)

-DISPLAY FUNCTION SPECIFIC (DB2) (DB2 Commands)

-DISPLAY GROUP (DB2) (DB2 Commands)

-DISPLAY LOCATION (DB2) (DB2 Commands)

-DISPLAY LOG (DB2) (DB2 Commands)

-DISPLAY PROCEDURE (DB2) (DB2 Commands)

-DISPLAY PROFILE (DB2) (DB2 Commands)

-DISPLAY RLIMIT (DB2) (DB2 Commands)

-DISPLAY THREAD (DB2) (DB2 Commands)

-DISPLAY TRACE (DB2) (DB2 Commands)

-DISPLAY UTILITY (DB2) (DB2 Commands)

Catalog tables

Help you determine when to reorganize table spaces and indexes.

Related information:

Deciding when to reorganize table spaces

Determining when to reorganize indexes

Maintaining DB2 database statistics

DB2 catalog tables (DB2 SQL)

Setting up your system for real-time statistics

EXPLAIN

Provides information about the access paths used by DB2. By using DB2 EXPLAIN you can capture and analyze information about plans, packages, or SQL statements when they are bound. The output appears in a supplied user table called *plan table* named PLAN_TABLE.

Additional EXPLAIN tables provide other kinds of information. For example, the *statement table*, named DSN_STATEMNT_TABLE, provides information about the estimated costs of executing SQL SELECT, INSERT, UPDATE, or DELETE statements. Similarly, the *function table*, named DSN_FUNCTION_TABLE contains information about user-defined functions that are called by a statement.

Many additional EXPLAIN tables exist to provide different types of information about the execution of SQL statements, and to enable the functions of certain optimization tools.

Users without authority to run EXPLAIN directly can obtain access path information for certain statements by calling the DB2-supplied EXPLAIN stored procedure (DSNAEXP). DSNAEXP is deprecated. Consider using the EXPLAIN privilege in place of DSNAEXP to explain statements for users that do not have the authority to execute EXPLAIN statements. DSNAEXP will be removed in a future release of DB2.

Related information:

Investigating SQL performance by using EXPLAIN

Working with and retrieving EXPLAIN table data

Interpreting data access by using EXPLAIN

EXPLAIN (DB2 SQL)

EXPLAIN tables

DSNAEXP stored procedure (DB2 SQL)

CICS journal

Facilities for creating and managing journals during CICS processing. Journals can contain any data the user needs to facilitate subsequent reconstruction of events or data changes.

CICS Attachment Facility statistics

Provide information about the use of CICS threads. This information can be displayed on a terminal or printed in a report.

Related information:

DSNC DISPLAY (CICS Transaction Server for z/OS)

CICS Transaction Server for z/OS DB2 Guide

CICS Monitor Facility

CICS monitoring collects data about the performance of all user- and CICS-supplied transactions during online processing for later offline analysis.

Related information:

CICS Monitor Facility (CICS Transaction Server for z/OS)

Real-time statistics stored procedure (DSNACCOX)

A sample stored procedure that makes recommendations to help you maintain your DB2 databases.

Related information:

DSNACCOX stored procedure (DB2 SQL)

Other tools and facilities for DB2 performance monitoring

IBM Data Studio

In addition to providing other administrative functions, IBM Data Studio provides query serviceability and tuning capabilities from workstation computers for queries that run in DB2 for z/OS environments.

Related information:

IBM Data Studio (Introduction to DB2 for z/OS)

IBM Data Server Manager

With IBM Data Server Manager, you can identify and analyze database problems by reviewing real-time and historical performance metrics. Tuning advisors provide expert recommendations about statistics, indexes, and performance improvement.

Related information:

IBM Data Server Manager (Introduction to DB2 for z/OS)

DB2 Connect

Can monitor and report DB2 server-elapsed time for client applications that access DB2 data.

Related information:

Reporting server-elapsed time

Generalized Trace Facility (GTF)

A z/OS service aid that can be used to collect DB2 trace data for performance analysis. GTF can also be used to analyze seek times and Supervisor Call instruction (SVC) usage, and for other services.

Related information:

Recording GTF trace data

Resource Measurement Facility™ (RMF)

An optional feature of z/OS that provides system-wide information on processor utilization, I/O activity, storage, and paging. RMF provides for three basic types of sessions: Monitor I, Monitor II, and Monitor III. Monitor I and Monitor II sessions collect and report data primarily about specific system activities. Monitor III sessions collect and report data about overall system activity in terms of work flow and delay.

Related information:

Monitoring system resources by using RMF

z/OS RMF User's Guide

Effective zSeries Performance Monitoring Using Resource Measurement Facility (IBM Redbooks)

Monitoring distributed processing with RMF

Group buffer pool monitoring with the coupling facility activity report of RMF (DB2 Data Sharing Planning and Administration)

System Management Facility (SMF)

A z/OS service aid used to collect information from various z/OS subsystems. SMF can be used to collect DB2 accounting, statistics, audit and performance trace data.

Related information:

Recording SMF trace data

Reporting data in SMF

Tivoli Decision Support for z/OS

A licensed program that collects SMF data into a DB2 database and allows you to create reports on the data.

Tivoli Decision Support for z/OS uses data from different sources, including SMF, the IMS log, the CICS journal, RMF, and DB2.

The data collection and reporting are based on user specifications. Therefore, experienced users can produce more useful reports than the predefined reports that are produced by other tools. Tivoli Decision Support also provides historical performance data that you can use to compare a current situation with previous data. Tivoli Decision Support is most useful for DB2 statistics and accounting records.

The following considerations apply to the use of Tivoli Decision Support for DB2 performance trace data:

- Because of the large number of different performance trace records, substantial effort is required to define the formats in Tivoli Decision Support. Changes in the records require review of the definitions.
- Tivoli Decision Support does not handle information from paired records, such as “start event” and “end event.” These record pairs are used by Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to calculate elapsed times, such as the elapsed time of I/Os and lock suspensions.

Therefore, the recommendation is to use Tivoli Decision Support only for accounting and statistics records in most cases, and to use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS for DB2 performance trace data. If you do not use Tivoli decision support at all, you can create user applications to extend Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS functions to provide reports for DB2 and QMF historical data.

Related information:

Tivoli Decision Support for z/OS

Tivoli OMEGAMON CICS Monitoring Facility (CMF)

Provides performance information about each CICS transaction executed. It can be used to investigate the resources used and the time spent processing transactions. Be aware that overhead is significant when CMF is used to gather performance information.

Related information:

CICS Monitor Facility (CICS Transaction Server for z/OS)

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

A performance monitoring tool that formats performance data. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS combines information from EXPLAIN and from the DB2 catalog. You can use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS for the following activities:

- Analysis of DB2 trace records: It displays information about the following objects and structures:
  - Access paths

  - Host variable definitions
  - Indexes
  - Join sequences
  - Lock types
  - Ordering
  - Plans
  - Packages
  - Tables
  - Table spaces
  - Table access sequences
- Reporting: The *batch report sets* present the data you select in comprehensive reports or graphs containing system-wide and application-related information for both single DB2 subsystems and DB2 members of a data sharing group. You can combine instrumentation data from several different DB2 locations into one report. Batch reports can be used to examine performance problems and trends over a period of time.
- Monitoring: The *online monitor* provides a current "snapshot" view of a running DB2 subsystem, including applications that are running. The history function displays information about subsystem and application activity in the recent past.
- Warehousing DB2 performance data. The *performance warehouse* function enables the following activities:
  - Saving DB2 trace and report data in a performance database for further investigation and trend analysis.
  - Configuring and scheduling the report and load processes from the workstation interface.
  - Defining and applying analysis functions to identify performance bottlenecks.
- Buffer pool analysis: The *buffer pool analyzer* function, which can help you to optimize buffer pool usage. When viewing the reports, you can:
  - Order by various identifiers such as buffer pool, plan, object, and primary authorization ID
  - Sort by getpage, sequential prefetch, and synchronous read
  - Filter the reports

  You can also simulate buffer pool usage for varying buffer pool sizes and analyze the results of the simulation reports to determine the impact of any changes before making those changes to your current system.
- End-to-end SQL monitoring: The *Extended Insight* feature tracks and measures the flow of a single SQL request through all its application components, from the database engine to the end-user. This feature is particularly useful for monitoring SQL performance in distributed environments.

Related information:

Tivoli OMEGAMON XE for Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS

Online monitoring and reporting (Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS)

Monitoring and problem determination (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

White Paper: OMEGAMON Extended Insight Analysis: Where is your application spending its time? (PDF)

End-to-End SQL Monitoring Workspace (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Quick Start Guide for the end-to-end SQL monitoring function (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

DB2 subsystem configuration, PE Client, and end-to-end SQL monitoring (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Tivoli Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS XE for DB2 Performance Monitor

Provides performance monitoring, reporting, and performance warehouse functions.

Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS performs a subset of the complete functions of Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS.

Related information:

Monitoring and problem determination (Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS)

DB2 Query Workload Tuner for z/OS

A full-featured SQL analysis and tuning tool, which enhances the functions provided by IBM Data Studio with additional capabilities and expert advisors. It also enables you to monitor and tune the performance of SQL queries and SQL query workloads.

Related information:

DB2 Query Workload Tuner for z/OS

IMS log records

Information about various IMS processes, including comprehensive information about transit times (actual system performance time), and IMS resource usage and availability.

Related information:

Understanding IMS log data (IMS Performance Analyzer)

IMS log records (IMS Performance Analyzer)

IMS Fast Path Log Analysis utility (DBFULTA0)

An IMS utility that provides performance reports for IMS Fast Path transactions.

Related information:

Fast Path Log Analysis utility (DBFULTA0)

IMS Monitor

The IMS Monitor collects data while the online IMS subsystem is running. It gathers information for all dispatch events and places it, in the form of IMS Monitor records, in a sequential data set.

Related information:

IMS Monitor

IMS Performance Analyzer for DB2 for z/OS
A performance analysis and tuning aid for Information Management System Database (IMS DB) and Transaction Manager (IMS TM) systems.

Related information:
IMS Performance analyzer

Monitoring CICS, and IMS

Certain facilities enable you to monitor the performance of DB2 in conjunction with CICS and IMS.

DB2 and CICS

To monitor DB2 and CICS, you can use the following facilities.

- RMF Monitor I and II for physical resource utilizations
- GTF for detailed I/O monitoring when needed
- Tivoli Decision Support for z/OS for application processor utilization, transaction performance, and system statistics.

You can use RMF Monitor II to dynamically monitor system-wide physical resource utilizations, which can show queuing delays in the I/O subsystem.

In addition, the CICS attachment facility DSNC DISPLAY command allows any authorized CICS user to dynamically display statistical information related to thread usage and situations when all threads are busy.

Be sure that the number of threads reserved for specific transactions or for the pool is large enough to handle the actual load. You can dynamically modify the value specified in the CICS resource definition online (RDO) attribute ACCOUNTREC with the DSNC MODIFY TRANSACTION command. You might also need to modify the maximum number of threads specified for the MAX USERS field on installation panel DSNTIPE.

DB2 and IMS

To monitor DB2 and IMS, you can use the following facilities.

- RMF Monitor I and II for physical resource utilizations
- GTF for detailed I/O monitoring when needed
- Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS IMS Performance Analyzer, or its equivalent, for response-time analysis and tracking all IMS-generated requests to DB2
- IMS Fast Path Log Analysis Utility (DBFULTA0) for performance reports for IMS Fast Path transactions.

The DB2 IMS attachment facility also allows you to use the DB2 command DISPLAY THREAD command to dynamically observe DB2 performance.

Related information:

DSNC DISPLAY (CICS Transaction Server for z/OS)

Monitoring tools for distributed environments

Certain monitoring tools are particularly useful for monitoring DB2 in distributed environments.

To monitoring DB2 performance in distributed environments:

- You can use the *Extended Insight* feature of Tivoli Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS XE for DB2 Performance Expert on z/OS provides end-to-end SQL monitoring.
- You can use InfoSphere Optim Query Workload Tuner to monitor and tune SQL statements and SQL workloads.

Related concepts:

Monitoring DB2 in distributed environments

DB2 subsystem configuration, PE Client, and end-to-end SQL monitoring (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

End-to-End SQL Monitoring Workspace (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related tasks:

"Monitoring threads and connections by using profiles" on page 108

Related reference:

DB2 Query Workload Tuner for z/OS

Related information:

Quick Start Guide for the end-to-end SQL monitoring function (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Chapter 41. Monitoring performance

Proactive performance monitoring is a key element of maintaining the health of your database management system.

Related tasks:

Investigating DB2 performance problems

Monitoring system resources by using RMF

You can monitor system resources to detect constraints for processor, I/O, and storage resources.

About this task

You can use the information that you gather for the following purposes to:

- Determine how resources, such as processor, I/O, and stage, are consumed in the system.
- Analyzing processor, I/O, and paging rates to detect bottlenecks in the system.
- Detect changes in the use of system resources over time, for comparable periods.

The following figure shows an example of a suggested system resources report.

```
                          SYSTEM RESOURCES REPORT        DATE xx/xx/xx
                                                         FROM xx:xx:xx
                                                         TO xx:xx:xx

TOTAL CPU Busy        74.3 %

  DB2 & IRLM           9.3 %
  IMS/CICS            45.3 %
  QMF Users            8.2 %
  DB2 Batch & Util     2.3 %
  OTHERS               9.2 %

  SYSTEM AVAILABLE    98.0 %

TOTAL I/Os/sec.       75.5

TOTAL Paging/sec.      6.8
                         Short          Medium         Long
                         Transaction    Transaction    Transaction

Average Response Time    3.2 secs       8.6 secs       15.0 secs

MAJOR CHANGES:
              DB2 application DEST07 moved to production
```

Figure 37. User-created system resources report

The RMF reports used to produce the information in the preceding figure were:

- The RMF CPU activity report, which lists TOTAL CPU busy and the TOTAL I/Os per second.
- RMF paging activity report, which lists the TOTAL paging rate per second for real storage.

- The RMF workload activity report, which is used to estimate where resources are spent. Each address space or group of address spaces to be reported on separately must have different SRM reporting or performance groups. The following SRM reporting groups are considered:
  - DB2 address spaces:

    DB2 database address space (*ssnm*DBM1)
    DB2 system services address space (*ssnm*MSTR)
    Distributed data facility (*ssnm*DIST)
    IRLM (IRLMPROC)
  - IMS or CICS
  - TSO-QMF
  - DB2 batch and utility jobs

  The CPU for each group is obtained using the ratio (A/B) × C, where:

  A is the sum of CPU and service request block (SRB) service units for the specific group
  B is the sum of CPU and SRB service units for all the groups
  C is the total processor utilization.

  The CPU and SRB service units must have the same coefficient.

  You can use a similar approach for an I/O rate distribution.

MAJOR CHANGES shows the important environment changes, such as:

- DB2 or any related software-level change
- DB2 changes in the load module for system parameters
- New applications put into production
- Increase in the number of QMF users
- Increase in batch and utility jobs
- Hardware changes

MAJOR CHANGES is also useful for discovering the reason behind different monitoring results.

Related concepts:

Major contributors to CPU time

DB2 in the z/OS environment (Introduction to DB2 for z/OS)

Related tasks:

Investigating CPU performance regression

Related reference:

z/OS RMF User's Guide

Monitoring transaction manager throughput

You can use IMS or CICS monitoring facilities to determine throughput, in terms of transactions processed, and transaction response times.

About this task

Depending on the transaction manager, you can use reports from the following tools and utilities:

- Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS IMS Performance Analyzer
- Fast Path Log Analysis Utility (DBFULTA0)
- Tivoli Decision Support for z/OS

In these reports:

- The transactions processed include DB2 and non-DB2 transactions.
- The transaction processor time includes the DB2 processor time for IMS but not for CICS.
- The transaction transit response time includes the DB2 transit time.

A historical database is useful for saving monitoring data from different periods. Such data can help you track the evolution of your system. You can use Tivoli Decision Support for z/OS or write your own application based on DB2 and QMF when creating this database.

Related concepts:

IMS Performance analyzer

Related information:

Fast Path Log Analysis utility (DBFULTA0)

Tivoli Decision Support for z/OS

Monitoring I/O and storage

You can monitor to the use of real and virtual storage by DB2.

Related tasks:

Managing I/O processing, response time, and throughput

Configuring storage for performance

Monitoring I/O activity of data sets

The best way to monitor your I/O activity against database data sets is through IFCID 0199 (statistics class 8).

About this task

IFCID 0199 provides information such as the average write I/O delay or the maximum delay for a particular data set over the last statistics reporting interval. The same information is also reported in the DISPLAY BUFFERPOOL command with the LSTATS option. Furthermore, Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports IFCID 0199 in batch reports and in online monitoring.

More detailed information is available, with more overhead, with the I/O trace (performance class 4). If you want information about I/O activity to the log and BSDS data sets, use performance class 5.

You can also use RMF to monitor data set activity. SMF record type 42-6 provides activity information and response information for a data set over a time interval. These time intervals include the components of I/O time, such as I/O disconnect time.Using RMF incurs about the same overhead as statistics class 8.

Related concepts:

Making I/O operations faster

Related tasks:

Controlling the number of I/O operations

Investigating class 3 suspension time

Related reference:

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

Monitoring and tuning buffer pools by using online commands

The DISPLAY BUFFERPOOL and ALTER BUFFERPOOL commands enable you to monitor and tune buffer pools on line, while DB2 is running, without the overhead of running traces.

Procedure

GUPI

To monitor and tune your buffer pools with online commands:

- Use the DISPLAY BUFFERPOOL command to display the current status of one or more active or inactive buffer pools.

  ```
  DISPLAY BUFFERPOOL(BP0) DETAIL
  ```

```
+DISPLAY BPOOL(BP2) DETAIL
DSNB401I + BUFFERPOOL NAME BP2, BUFFERPOOL ID 2, USE COUNT 47
DSNB402I + BUFFER POOL SIZE             = 4000
           AUTOSIZE                     = NO
           ALLOCATED      = 4000        TO BE DELETED    = 0
DSNB404I + THRESHOLDS -
             VP SEQUENTIAL         = 80
             DEFERRED WRITE        = 85   VERTICAL DEFERRED WRT = 80,  0
             PARALLEL SEQUENTIAL   = 50   ASSISTING PARALLEL SEQ = 0
DSNB406I + PGFIX ATTRIBUTE -
            CURRENT  =  NO
            PENDING  =  NO
            PAGE STEALING METHOD = LRU
DSNB409I + INCREMENTAL STATISTICS SINCE 14:57:55 JAN 22, yyyy
DSNB411I + RANDOM GETPAGE    =   491222 SYNC READ I/O (R) =    18193
           SEQ.   GETPAGE    =  1378500 SYNC READ I/O (S) =        0
           DMTH HIT          =        0 PAGE-INS REQUIRED =   460400
           SEQUENTIAL        =      200 VPSEQT HIT       =        0
           RECLASSIFY        =        0
DSNB412I + SEQUENTIAL PREFETCH
             REQUESTS        =    41800   PREFETCH I/O   =    14473
             PAGES READ      =   444030
DSNB413I + LIST PREFETCH -
             REQUESTS        =     9046   PREFETCH I/O    =     2263
             PAGES READ      =     3046
DSNB414I + DYNAMIC PREFETCH -
             REQUESTS        =     6680   PREFETCH I/O    =      142
             PAGES READ      =     1333
DSNB415I + PREFETCH DISABLED -
             NO BUFFER       =        0   NO READ ENGINE  =        0
DSNB420I + SYS PAGE UPDATES  =   220425 SYS PAGES WRITTEN =    35169
           ASYNC WRITE I/O   =     5084 SYNC WRITE I/O    =        3
           PAGE-INS REQUIRED =       45
DSNB421I + DWT HIT           =        2 VERTICAL DWT HIT  =        0
DSNB440I + PARALLEL ACTIVITY -
           PARALLEL REQUEST  =        0 DEGRADED PARALLEL   =      0
DSNB441I + LPL ACTIVITY -
           PAGES ADDED       =        0
DSN9022I + DSNB1CMD '+DISPLAY BPOOL' NORMAL COMPLETION
```

Figure 38. Sample output from the DISPLAY BUFFERPOOL command

In figure above, find the following fields:

  - `SYNC READ I/O (R)` shows the number of random synchronous read I/O operations. `SYNC READ I/O (S)` shows the number of sequential synchronous read I/O operations. Sequential synchronous read I/Os occur when prefetch is disabled.

    To determine the total number of synchronous read I/Os, add SYNC READ I/O (S) and SYNC READ I/O (R).
  - In message DSNB412I, `REQUESTS` shows the number of times that sequential prefetch was triggered, and `PREFETCH I/O` shows the number of times that sequential prefetch occurred. `PAGES READ` shows the number of pages read using sequential prefetch.
  - `SYS PAGE UPDATES` corresponds to the number of buffer updates.
  - `SYS PAGES WRITTEN` is the number of pages written to disk.
  - `DWT HIT` is the number of times the deferred write threshold (DWQT) was reached. This number is workload dependent.
  - `VERTICAL DWT HIT` is the number of times the vertical deferred write threshold (VDWQT) was reached. This value is per data set, and it is related to the number of asynchronous writes.
- Use the LSTATS option of the DISPLAY BUFFERPOOL command to obtain buffer pool information on a specific data set. For example, you can use the LSTATS option to:
  - Provide page count statistics for a certain index. With this information, you could determine whether a query used the index in question, and perhaps drop the index if it was not used.
  - Monitor the response times on a particular data set. If you determine that I/O contention is occurring, you could redistribute the data sets across your available disks.

  This same information is available with IFCID 0199 (statistics class 8).
- Use the ALTER BUFFERPOOL command to change the following attributes:
  - Buffer pool size: VPSIZE
  - Thresholds:
    - VPSEQT
    - VPPSEQT
    - VPXPSEQT
    - DWQT
    - VDWQT
  - Page-stealing algorithm: PGSTEAL
  - Page fix attribute: PGFIX
  - Automatic adjustment attribute: AUTOSIZE

Example

Because the number of synchronous read I/O is relatively high, you might tune the buffer pools by changing the buffer pool specifications. For example, you might increase the buffer pool size to reduce the amount of unnecessary I/O, which would make buffer operations more efficient. To do that, you would enter the following command:

```
-ALTER BUFFERPOOL(BP0) VPSIZE(nnnn)
```

GUPI

What to do next

Buffer Pool Analyzer: You can use the Buffer Pool Analyzer for z/OS to get recommendations buffer pool allocation changes and to do “what if” analysis of your buffer pools.

Related tasks:

Tuning database buffer pools

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

DB2 Buffer Pool Analyzer for z/OS

Related information:

DSNB401I (DB2 Messages)

The buffer pool hit ratio

Buffer pool hit ratio is a measure of how often a page access (a getpage) is satisfied without requiring an I/O operation.

PSPI

You can help some of your applications and queries by making the buffer pools large enough to increase the buffer hit ratio.

Accounting reports, which are application related, show the average hit ratio for multiple occurrences of applications or threads. An accounting trace report shows the hit ratio for a single application or thread. The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS buffer pool statistics report shows the hit ratio for the subsystem as a whole. For example, the buffer-pool hit ratio is shown in the `BPOOL HIT RATIO (%)` field in the following figure.

```
TOT4K  READ OPERATIONS       QUANTITY  TOT4K  WRITE OPERATIONS      QUANTITY
---------------------------  --------  ---------------------------  --------
BPOOL HIT RATIO (%)             73.12  BUFFER UPDATES                 220.4K
                                       PAGES WRITTEN                35169.00
GETPAGE REQUEST               1869.7K  BUFF.UPDATES/PAGES WRITTEN       6.27
GETPAGE REQUEST-SEQUENTIAL    1378.5K
GETPAGE REQUEST-RANDOM         491.2K  SYNCHRONOUS WRITES               3.00
                                       ASYNCHRONOUS WRITES           5084.00
SYNCHRONOUS READS            54187.00
SYNCHRON. READS-SEQUENTIAL   35994.00  PAGES WRITTEN PER WRITE I/O      5.78
SYNCHRON. READS-RANDOM       18193.00
                                       HORIZ.DEF.WRITE THRESHOLD        2.00
GETPAGE PER SYN.READ-RANDOM     27.00  VERTI.DEF.WRITE THRESHOLD        0.00
                                       DM THRESHOLD                     0.00
SEQUENTIAL PREFETCH REQUEST  41800.00  WRITE ENGINE NOT AVAILABLE       0.00
SEQUENTIAL PREFETCH READS    14473.00  PAGE-INS REQUIRED FOR WRITE     45.00
PAGES READ VIA SEQ.PREFETCH    444.0K
S.PRF.PAGES READ/S.PRF.READ     30.68

LIST PREFETCH REQUESTS        9046.00
LIST PREFETCH READS           2263.00
PAGES READ VIA LST PREFETCH   3046.00
L.PRF.PAGES READ/L.PRF.READ      1.35

DYNAMIC PREFETCH REQUESTED    6680.00
DYNAMIC PREFETCH READS         142.00
PAGES READ VIA DYN.PREFETCH   1333.00
D.PRF.PAGES READ/D.PRF.READ      9.39
PREF.DISABLED-NO BUFFER          0.00
PREF.DISABLED-NO READ ENG        0.00

PAGE-INS REQUIRED FOR READ     460.4K
```

Figure 39. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS database buffer pool statistics (modified)

The buffer hit ratio uses the following formula to determine how many getpage operations did not require an I/O operation:

```
Hit ratio = ((getpage-requests - pages-not-in-bp - pages-read-from-disk) /
            (getpage-requests - pages-not-in-bp)) * 100
```

The formula uses the following values, from fields in the buffer pool statistics report:

- *getpage-requests* is the number of getpage requests, from theGETPAGE REQUEST field.
- *pages-not-in-bp* is the number of getpage requests that failed because the page was not in the buffer pool, from the COND. REQUEST - NOT FOUND field.
- *pages-read-from-disk* is the sum of the values from the following fields:

  SYNCHRONOUS READS: Number of synchronous reads

  PAGES READ VIA SEQ.PREFETCH: Number of pages read through sequential prefetch.

  PAGES READ VIA LST PREFETCH: Number of pages read through list prefetch.

  PAGES READ VIA DYN.PREFETCH: Number of pages read through dynamic prefetch.

If there are 1000 getpage requests, all requested pages are in the buffer pool, and 100 pages are read from disk, the equation is as follows:

```
Hit ratio = ((1000 - 0 - 100)/1000)) * 100
```

The hit ratio in this case is 90%.

Highest and lowest hit ratios

Highest hit ratio

The highest possible value for the hit ratio is 1.0 (for 100%), which is achieved when every page requested is always in the buffer pool. Reading index non-leaf pages tend to have a very high hit ratio since they are frequently re-referenced and thus tend to stay in the buffer pool.

Lowest hit ratio

The lowest hit ratio occurs when the requested page is not in the buffer pool; in this case, the hit ratio is 0 or less. A negative hit ratio means that prefetch has brought pages into the buffer pool that are not subsequently referenced. The pages are not referenced because either the query stops before it reaches the end of the table space or DB2 must take the pages away to make room for newer ones before the query can access them.

A low hit ratio is not always bad

While it might seem desirable to make the buffer hit ratio as close to 100% as possible, do not automatically assume a low buffer-pool hit ratio is bad. The hit ratio is a relative value, based on the type of application. For example, an application that browses huge amounts of data using table space scans might very well have a buffer-pool hit ratio of 0, or possibly even a negative number because of prefetch processing. What you want to watch for is those cases where the hit ratio drops significantly for the same application. In those cases, it might be helpful to investigate further.

Buffer Pool hit ratio is not meaningful for buffer pools that have high number of sequentially accessed objects. You can move sequentially accessed objects to a separate buffer pool to improve buffer pool hit ratio for randomly accessed objects.

Using OMEGAMON to monitor buffer pool statistics

You can find information about buffer pools in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report.

Procedure

PSPI

To analyze your buffer pools with Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS:

1. Examine the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS statistics report.
2. Increase the size of the buffer pool in the following situations:
   - Sequential prefetch is inhibited. `PREF.DISABLED-NO BUFFER` shows how many times sequential prefetch is disabled because the sequential prefetch threshold (90% of the pages in the buffer pool are unavailable) has been reached.
   - You detect poor update efficiency. You can determine update efficiency by checking the values in both of the following fields:
     - `BUFF.UPDATES/PAGES WRITTEN`
     - `PAGES WRITTEN PER WRITE I/O`

     In evaluating the values you see in these fields, remember that no values are absolutely acceptable or absolutely unacceptable. Each installation's workload is a special case. To assess the update efficiency of your system, monitor for overall trends rather than for absolute high values for these ratios.

The following factors impact buffer updates per pages written and pages written per write I/O:

– Sequential nature of updates

– Number of rows per page

– Row update frequency

For example, a batch program that processes a table in skip sequential mode with a high row update frequency in a dedicated environment can achieve very good update efficiency. In contrast, update efficiency tends to be lower for transaction processing applications, because transaction processing tends to be random.

The following factors affect the ratio of pages written per write I/O:

Checkpoint frequency
The checkpoint frequency values specify the intervals for DB2 system checkpoints. The CHCKTYPE subsystem parameter controls whether the system checkpoint interval is based on the number of log records, a time interval, or both. The CHKFREQ subsystem parameter contains the interval value when only one type is specified. If both time-based and log based intervals are specified, the values of the CHKLOGR and CHKMINS subsystem parameters define the intervals.

Frequency of active log switch
DB2 takes a system checkpoint each time the active log is switched. If the active log data sets are too small, checkpoints occur often, which prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O.

Buffer pool size
The deferred write thresholds (VDWQT and DWQT) are a function of buffer pool size. If the buffer pool size is decreased, these thresholds are reached more frequently, causing I/Os to be scheduled more often to write some of the pages on the deferred write queue to disk. This prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O.

Number of data sets, and the spread of updated pages across them
The maximum number of pages written per write I/O is 32, subject to a limiting scope of 180 pages (roughly one cylinder).

Example: If your application updates page 2 and page 179 in a series of pages, the two changed pages could potentially be written with one write I/O. But if your application updates page 2 and page 185 within a series of pages, writing the two changed pages would require two write I/Os because of the 180-page limit. Updated pages are placed in a deferred write queue based on the data set. For batch processing it is possible to achieve a high ratio of pages written per write I/O, but for transaction processing the ratio is typically lower.

For LOAD, REORG, and RECOVER, the maximum number of pages written per write I/O is 64, and the scope is typically unlimited. However, in some cases, such as loading a partitioned table space with nonpartitioned indexes, a 180-page limit scope exists.

- The SYNCHRONOUS WRITES field contains a large value. This field represents the total number of immediate writes. Synchronous or *immediate writes* occur in the following situations:
  - The immediate write threshold is reached.
  - No deferred write engines are available.
  - More than two checkpoints pass before a page is written.

  DB2 sometimes uses immediate writes even when the data manager and immediate write thresholds are not exceeded, such as when more than two checkpoints pass before a page is written. That type of situation does not indicate a buffer pool shortage.

  If a large number of synchronous writes occur, monitor the value of the DM Critical Threshold Reached field. You can ignore the value in theSYNCHRONOUS WRITES field when DM Critical Threshold Reached value is zero. Reaching the data manager threshold implies that the lower immediate write threshold was crossed. Otherwise, increase the size of the buffer pool.
- The data management threshold is reached.The extra increased getpage and release requests that result from reaching the data manager threshold increase CPU usage. The DM THRESHOLD field shows how many times a page was immediately released because the data management threshold was reached. The counter is incremented whenever 95% or more of the buffer pool was filled with updated pages. The quantity listed for this field should be zero. If a high number is shown, increase the buffer pool size. Reducing the buffer pool deferred write thresholds might also help.

Also note the following fields:

WRITE ENGINE NOT AVAILABLE

Records the number of times that asynchronous writes were deferred because DB2 reached its maximum number of concurrent writes. You cannot change this maximum value. This field has a nonzero value occasionally.

You can reduce the value of the deferred write thresholds to reduce the intensity of deferred write spikes. If response times are long, the I/O subsystem might require tuning.

PREF.DISABLED-NO READ ENG

Records the number of times that a sequential prefetch was not performed because the maximum number of concurrent sequential prefetch operations was reached. Instead, synchronous reads were used. You cannot change this maximum value. If the I/O response times are long, the I/O subsystem might required tuning.

PAGE-INS REQUIRED FOR WRITE and PAGE-INS REQUIRED FOR READ

Records the number of page-ins that are required for a read or write I/O. When the buffer pools are first allocated, the count might be high. After the first allocations are complete, the count should be close to zero. Page-ins for read might also occur if the buffer pool AUTOSIZE option is used to enable dynamic expansion of the buffer pool. If the counts are high during steady state, check MVS paging. If excessive MVS paging occurs, allocate more real storage to the LPAR.

Example

The following figure shows where you can find important values in the statistics report.

```
TOT4K  READ OPERATIONS       QUANTITY  TOT4K  WRITE OPERATIONS      QUANTITY
---------------------------  --------  ---------------------------  --------
BPOOL HIT RATIO (%)             73.12  BUFFER UPDATES                 220.4K
                                       PAGES WRITTEN                35169.00
GETPAGE REQUEST               1869.7K  BUFF.UPDATES/PAGES WRITTEN       6.27
GETPAGE REQUEST-SEQUENTIAL    1378.5K
GETPAGE REQUEST-RANDOM         491.2K  SYNCHRONOUS WRITES               3.00
                                       ASYNCHRONOUS WRITES           5084.00
SYNCHRONOUS READS            54187.00
SYNCHRON. READS-SEQUENTIAL   35994.00  PAGES WRITTEN PER WRITE I/O      5.78
SYNCHRON. READS-RANDOM       18193.00
                                       HORIZ.DEF.WRITE THRESHOLD        2.00
GETPAGE PER SYN.READ-RANDOM     27.00  VERTI.DEF.WRITE THRESHOLD        0.00
                                       DM THRESHOLD                     0.00
SEQUENTIAL PREFETCH REQUEST  41800.00  WRITE ENGINE NOT AVAILABLE       0.00
SEQUENTIAL PREFETCH READS    14473.00  PAGE-INS REQUIRED FOR WRITE     45.00
PAGES READ VIA SEQ.PREFETCH    444.0K
S.PRF.PAGES READ/S.PRF.READ     30.68

LIST PREFETCH REQUESTS        9046.00
LIST PREFETCH READS           2263.00
PAGES READ VIA LST PREFETCH   3046.00
L.PRF.PAGES READ/L.PRF.READ      1.35

DYNAMIC PREFETCH REQUESTED    6680.00
DYNAMIC PREFETCH READS         142.00
PAGES READ VIA DYN.PREFETCH   1333.00
D.PRF.PAGES READ/D.PRF.READ      9.39
PREF.DISABLED-NO BUFFER          0.00
PREF.DISABLED-NO READ ENG        0.00

PAGE-INS REQUIRED FOR READ     460.4K
```

Figure 40. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS database buffer pool statistics (modified)

Related concepts:

Buffer pool thresholds

Related tasks:

Tuning database buffer pools

Using Buffer Pool Analyzer (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Choosing a checkpoint frequency

Related reference:

-ALTER BUFFERPOOL (DB2) (DB2 Commands)

DB2 Buffer Pool Analyzer for z/OS

CHECKPOINT TYPE field (CHKTYPE subsystem parameter) (DB2 Installation and Migration)

RECORDS/CHECKPOINT field (CHKFREQ and CHKLOGR subsystem parameters) (DB2 Installation and Migration)

MINUTES/CHECKPOINT field (CHKFREQ and CHKMINS subsystem parameters) (DB2 Installation and Migration)

Monitoring work file data sets

You should monitor how work files use devices, both in terms of space use and I/O response times.

About this task

Work file data sets are used for sorting, for materializing views and nested table expressions, for temporary tables, and for other activities. DB2 does not distinguish or place priorities on these uses of the work file data sets. Excessive activity from one type of use can detract from the performance of others.

When a temporary table is populated using an INSERT statement, it uses work file space.

No other process can use the same work file space as that temporary work file table until the table goes away. The space is reclaimed when the application process commits or rolls back, or when it is deallocated, depending which RELEASE option was used when the plan or package was bound.

Procedure

To monitor work file usage by temporary tables:

- Keep work files in a separate buffer pool.
- Run a performance class 8 trace. IFCID 0311 data can be used to monitor the use of declared temporary tables.

Related tasks:

Calculating sort pool size (DB2 Installation and Migration)

Monitoring catalog statistics

You can improve access path selection for SQL statements by monitoring the health of statistics for database objects. The health of these statistics is of critical importance for efficient access paths for SQL statements.

About this task

PSPI

DB2 uses a cost-based optimizer, which makes decisions based on the statistics that are available for the tables and indexes. When accurate statistics are not available for database objects, the cost estimates that DB2 relies on for choosing the most efficient access path become incorrect.

Procedure

To monitor the health of statistics for database objects, use one or more of the following approaches:

- Call the DSNACCOX stored procedure to discover whether to invoke the REORG or RUNSTATS utilities for database objects. The DSNACCOX stored procedures uses real-time statistics values to recommend activities for maintaining data organization and statistics for database objects.
- Specify the REPORT YES option when you invoke the RUNSTATS utility.
- Issue the SELECT statement to query the catalog statistics. You can find a set of sample SELECT statements for querying statistics information from the catalog in member DSNTESP of the SDSNSAMP data set.

What to do next

If the statistics in the DB2 catalog no longer correspond to the organization and content of your data (or the DSNACCOX stored procedure recommends), take the following actions:

1. Run the REORG utility to reorganize the data.
2. Run the RUNSTATS utility to collect accurate statistics for database objects.
3. Rebind the applications that use static SQL statements so that DB2 can choose the most efficient access paths.

PSPI

Related tasks:

Maintaining data organization and statistics

Automating statistics maintenance

Investigating access path problems

Related reference:

DB2 catalog tables (DB2 SQL)

Statistics used for access path selection

DSNACCOX stored procedure (DB2 SQL)

RUNSTATS (DB2 Utilities)

REORG INDEX (DB2 Utilities)

REORG TABLESPACE (DB2 Utilities)

Monitoring concurrency and locks

You can monitor the use of locks to improve concurrency and prevent problems such as contention, suspensions, timeouts, or deadlocks.

Procedure

PSPI

To monitor the use of locks by DB2, use any of the following approaches:

- Always run statistics classes 1, 3, and 4 and accounting classes 1 and 3. The statistics reports provide counters for timeouts, deadlocks and suspensions. Statistics class 3 includes IFCID 0172 (deadlocks) and IFCID 0196 (deadlocks). If deadlocks or timeouts are a concern, look at these detail records to investigate the situation during exceptional situations.
- Use the accounting reports. The accounting reports show the locking activity under the heading of LOCKING. The other key indication of locking problems are the class 3 suspensions LOCK/LATCH(DB2+IRLM). If locking and latching are increasing the elapsed time of your transactions or batch work, you might want to investigate further.
- Use the statistics trace to monitor the system-wide use of locks, the accounting trace to monitor locks used by a particular application process.
- Use EXPLAIN to monitor the locks required by a particular SQL statement, or all the SQL in a particular plan or package.

- Use performance trace classes 1, 2, 3, 6, and 7 and analyze the SQL and locking trace data. You can use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to generate SQL activity, locking, and record trace reports.

Related concepts:

Locks and Latches (DB2 for z/OS Best Practices)

Related tasks:

Analyzing concurrency

Improving concurrency

Configuring subsystems for concurrency

Designing databases for concurrency

Programming for concurrency

Related reference:

SQL Activity Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Locking Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Record Trace Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Monitoring locks by using statistics and accounting traces

You can use statistics and accounting traces to monitor locking activity.

About this task

PSPI

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS provides one way to view the trace results. The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports for the accounting and statistics traces each corresponds to a single DB2 trace record.

Procedure

To monitor locking activity by using statistic and accounting reports:

- Use the statistic trace to monitor locking activity on the subsystem. The following figure shows a portion of the statistics trace. It shows how many suspensions, timeouts, deadlocks, and lock escalations occur in the trace record. You can also use the statistics trace to monitor lock escalation.

```
LOCKING,ACTIVITY            QUANTITY  /SECOND  /THREAD  /COMMIT
--------------------------  --------  -------  -------  -------
SUSPENSIONS (ALL)             107.4K    30.33     0.17     0.07
SUSPENSIONS (LOCK ONLY)      1126.00     0.32     0.00     0.00
SUSPENSIONS (IRLM LATCH)    70701.00    19.97     0.11     0.05
SUSPENSIONS (OTHER)         35524.00    10.04     0.06     0.02

TIMEOUTS                        0.00     0.00     0.00     0.00
DEADLOCKS                       0.00     0.00     0.00     0.00

LOCK REQUESTS               36558.4K    10.3K    57.12    25.30
UNLOCK REQUESTS              9007.3K  2544.45    14.07     6.23
QUERY REQUESTS              34751.00     9.82     0.05     0.02
CHANGE REQUESTS               853.4K   241.08     1.33     0.59
OTHER REQUESTS                  0.00     0.00     0.00     0.00

LOCK ESCALATION (SHARED)        2.00     0.00     0.00     0.00
LOCK ESCALATION (EXCLUSIVE)     1.00     0.00     0.00     0.00

DRAIN REQUESTS               6131.00     1.73     0.01     0.00
DRAIN REQUESTS FAILED          58.00     0.02     0.00     0.00
CLAIM REQUESTS              21397.8K  6044.59    33.43    14.81
```

Figure 41. Locking activity blocks from statistics trace

- Try to keep the number of unlock requests per commit to fewer than five. You can expect a greater number of unlock requests per commit for applications that use the ISOLATION(CS) and CURRENTDATA(YES) bind options.
- Use accounting trace data to investigate specific applications that have high numbers of lock and unlock requests. The following figure shows a portion of the accounting trace. It shows the suspensions, timeouts, deadlocks, lock escalations for a particular application. It also shows the maximum number of concurrent page locks that are held and acquired during the trace. If this value is large, you can try to reduce it by reducing the number of locks that are acquired, or by committing more frequently. This value is also the basis for setting the values of the NUMLKUS and NUMLKTS subsystem parameters.

  The LOCK/LATCH(DB2_IRLM) time is broken down into specific values for the IRLM locks and latches and DB2 latch activity. The accounting report also breaks down the total suspensions in lock, lock, latch, and other suspensions. You can use this information to isolate the problem.

```
CLASS 3 SUSPENSIONS   ELAPSED TIME    EVENTS
--------------------  ------------  --------
LOCK/LATCH(DB2+IRLM)      0.362856      1947
 IRLM LOCK+LATCH          0.274047       692
 DB2 LATCH                0.088809      1455
............................................

LOCKING               TOTAL
------------------- --------
TIMEOUTS                   0
DEADLOCKS                  0
ESCAL.(SHAR)               0
ESCAL.(EXCL)               0
MAX PG/ROW LCK HELD       13
LOCK REQUEST          407685
UNLOCK REQST           59302
QUERY REQST                0
CHANGE REQST              10
OTHER REQST                0
TOTAL SUSPENSIONS        692
 LOCK SUSPENS             60
 IRLM LATCH SUSPENS      632
 OTHER SUSPENS             0
```

Figure 42. Locking activity blocks from the accounting trace report

Related concepts:

Statistics trace

Accounting trace

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related tasks:

Improving concurrency

Choosing an ISOLATION option

Choosing a CURRENTDATA option

Related reference:

Lock monitoring with the DB2 accounting trace (DB2 Data Sharing Planning and Administration)

Locking in the accounting report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Statistics Report and Trace Blocks (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

LOCKS PER USER field (NUMLKUS subsystem parameter) (DB2 Installation and Migration)

LOCKS PER TABLE(SPACE) field (NUMLKTS subsystem parameter) (DB2 Installation and Migration)

Using EXPLAIN to identify locks chosen by DB2

You can use DB2 EXPLAIN to determine what kind of locks DB2 uses to process an SQL statement.

Procedure

To analyze the locks used by DB2:

1. Use the EXPLAIN statement, or the EXPLAIN option of the BIND and REBIND sub-commands, to determine which modes of table, partition, and table space locks DB2 initially assigns for an SQL statement. EXPLAIN stores the results in the PLAN_TABLE.
2. Query the PLAN_TABLE. Each PLAN_TABLE row describes the processing for a single table, either one named explicitly in the SQL statement that is being explained or an intermediate table that DB2 creates. The TSLOCKMODE column shows the initial lock mode for that table.

   The lock mode value applies to the table or the table space, depending on the value of LOCKSIZE and whether the table space is segmented or nonsegmented. For partitioned and universal table spaces, the lock mode applies only to locked partitions. Lock modes for LOB and XML table spaces are not reported with EXPLAIN.

   The following tables show the table or table space locks that DB2 chooses, and whether page or row locks are used also, for each particular combination of lock mode and size.

Universal table spaces:

EXPLAIN lock type	IS	S	IX	U	X
Table space lock acquired is:	IS	S	IX	U	X
Page or row locks acquired?	Yes	No	Yes	No	No
Mass delete locks acquired?	No	No	No	No	No

Non-segmented

EXPLAIN lock type	IS	S	IX	U	X
Table space lock acquired is:	IS	S	IX	U	X
Page or row locks acquired?	Yes	No	Yes	No	No
Mass delete locks acquired?	Yes	Yes	No	No	No

Segmented table spaces with LOCKSIZE ANY, ROW, or PAGE

EXPLAIN lock type	IS	S	IX	U	X
Table space lock acquired is:	IS	IS	IX	n/a	IX
Table lock acquired is:	IS	S	IX	n/a	X
Page or row locks acquired?	Yes	No	Yes	n/a	No
Mass delete locks acquired?	Yes	Yes	No	No	No

Segmented table spaces with LOCKSIZE TABLE

EXPLAIN lock type	IS	S	IX	U	X
Table space lock acquired is:	n/a	IS	n/a	IX	IX
Table lock acquired is:	n/a	S	n/a	U	X
Page or row locks acquired?	n/a	No	n/a	No	No
Mass delete locks acquired?	Yes	Yes	No	No	No

Segmented table spaces with LOCKSIZE TABLESPACE

EXPLAIN lock type	IS	S	IX	U	X
Table space lock acquired is:	n/a	S	n/a	U	X
Table lock acquired is:	n/a	n/a	n/a	n/a	n/a
Page or row locks acquired?	n/a	No	n/a	No	No
Mass delete locks acquired?	Yes	Yes	No	No	No

Related concepts:

Investigating SQL performance by using EXPLAIN

Interpreting data access by using EXPLAIN

Locks for LOB data

Locks for XML data

Related tasks:

Analyzing concurrency

Improving concurrency

Related reference:

PLAN_TABLE

Deadlock detection scenarios

The following deadlock scenarios demonstrate deadlock detection and use of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS deadlock detail report to determine the cause of a deadlock.

PSPI

Deadlocks can often be avoided if concurrent applications access data in the same sequence. In some cases row-level locking can reduce the number of deadlocks and timeouts.

The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Locking Trace - Deadlock report formats the information contained in trace record IFCID 0172 (statistics class 3). The report outlines all the resources and agents involved in a deadlock and the significant locking parameters, such as lock state and duration, related to their requests.

These examples assume that statistics class 3 and performance class 1 are activated. Performance class 1 is activated to get IFCID 105 records, which contain the translated names for the database ID and the page set OBID.

The scenarios that follow use three of the DB2 sample tables, DEPT, PROJ, and ACT. They are all defined with LOCKSIZE ANY. Indexes are used to access all three tables. As a result, contention for locks is only on data pages.

PSPI

Related concepts:

Lock size

Related tasks:

Improving concurrency

Monitoring concurrency and locks

Related reference:

CREATE TABLESPACE (DB2 SQL)

ALTER TABLESPACE (DB2 SQL)

Related information:

"Deadlock" on page 189

Scenario: Two-way deadlock with two resources

When two agents contend for resources, the result is a deadlock in which one of the agents is rolled back.

PSPI

Two transactions and two resources are involved. First, transaction LOC2A acquires a lock on one resource while transaction LOC2B acquires a lock on another. Next, the two transactions each request locks on the resource held by the other.

The transactions execute in the following order:

LOC2A

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

LOC2B

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 8.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC2A obtains a U lock on page 2 in table DEPT, to open its cursor for update.

2. LOC2B obtains a U lock on a page 8 in table PROJ, to open its cursor for update.
3. LOC2A attempts to access page 8, to open its cursor but cannot proceed because of the lock held by LOC2B.
4. LOC2B attempts to access page 2, to open its cursor but cannot proceed because of the lock held by LOC2A.

DB2 selects one of the transactions and rolls it back, releasing its locks. That allows the other transaction to proceed to completion and release its locks also.

The following figure shows the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Locking Trace - Deadlock report that is produced for this situation.

The report shows that the only transactions involved came from plans LOC2A and LOC2B. Both transactions came in from BATCH.

```
⋮
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP           --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
SYSADM   RUNLOC2A TSO          20:32:30.68850025 DEADLOCK                                   COUNTER =    2      WAITERS =    2
SYSADM   'BLANK'  AADD32FD8A8C N/P                                                        TSTAMP  =04/02/95 20:32:30.68
LOC2A    BATCH                                            DATAPAGE  DB  =DSN8D42A           HASH    =X'01060304'
    A                                                               OB  =DEPT               ---------------- BLOCKER IS HOLDER -----
                                                                    PAGE=X'000002'          LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2A      CORRID=RUNLOC2A
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            STATE   =U
                                                                                            ---------------- WAITER ----------------
                                                                                            LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2B      CORRID=RUNLOC2B
                                                                                            DURATION=MANUAL     PRIMAUTH=KATHY
                                                                                            REQUEST =LOCK       WORTH   =    18
                                                                                            STATE   =U

                                                          DATAPAGE  DB  =DSN8D42A           HASH    =X'01060312'
                                                                    OB  =PROJ               ---------------- BLOCKER IS HOLDER -----
                                                                    PAGE=X'000008'          LUW='BLANK'.EGTVLU2.AA65FEDC1022
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2B      CORRID=RUNLOC2B
                                                                                            DURATION=MANUAL     PRIMAUTH=KATHY
                                                                                            STATE   =U
                                                                                            ---------------- WAITER -------*VICTIM*-
                                                                                            LUW='BLANK'.EGTVLU2.AADD32FD8A8C
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC2A      CORRID=RUNLOC2A
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            REQUEST =LOCK       WORTH   =   17
                                                                                            STATE   =U
```

Figure 43. Deadlock scenario 1: Two transactions and two resources

The lock held by transaction 1 (LOC2A) is a data page lock on the DEPT table and is held in U state. (The value of MANUAL for duration means that, if the plan was bound with isolation level CS and the page was not updated, then DB2 is free to release the lock before the next commit point.)

Transaction 2 (LOC2B) was requesting a lock on the same resource, also of mode U and hence incompatible.

The specifications of the lock held by transaction 2 (LOC2B) are the same. Transaction 1 was requesting an incompatible lock on the same resource. Hence, the deadlock.

Finally, note that the entry in the trace, identified at A, is LOC2A. That is the selected thread (the “victim”) whose work is rolled back to let the other proceed.

PSPI

Related tasks:

Improving concurrency

Monitoring concurrency and locks

Related information:

“Deadlock” on page 189

Scenario: Three-way deadlock with three resources

When three agents contend for resources, the result is a deadlock in which one of the agents is rolled back. Three transactions and three resources are involved.

PSPI

First, the three transactions each acquire a lock on a different resource. LOC3A then requests a lock on the resource held by LOC3B, LOC3B requests a lock on the resource held by LOC3C, and LOC3C requests a lock on the resource held by LOC3A.

The transactions execute as follows:

LOC3A

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

LOC3B

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on ACT and fetch from page 6.
3. Update page 6.
4. Update page 8.
5. Close both cursors and commit.

LOC3C

1. Declare and open a cursor for update on ACT and fetch from page 6.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 6.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC3A obtains a U lock on page 2 in DEPT, to open its cursor for update.
2. LOC3B obtains a U lock on page 8 in PROJ, to open its cursor for update.
3. LOC3C obtains a U lock on page 6 in ACT, to open its cursor for update.
4. LOC3A attempts to access page 8 in PROJ but cannot proceed because of the lock held by LOC3B.
5. LOC3B attempts to access page 6 in ACT cannot proceed because of the lock held by LOC3C.
6. LOC3C attempts to access page 2 in DEPT but cannot proceed, because of the lock held by LOC3A.

DB2 rolls back LOC3C and releases its locks. That allows LOC3B to complete and release the lock on PROJ so that LOC3A can complete. LOC3C can then try again.

The following figure shows the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Locking Trace - Deadlock report produced for this situation.

PSPI

```
:
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP            --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
SYSADM   RUNLOC3C TSO          15:10:39.33061694 DEADLOCK                                   COUNTER =    3      WAITERS =    3
SYSADM   'BLANK'  AADE2CF16F34 N/P                                                          TSTAMP  =04/03/95 15:10:39.31
LOC3C    BATCH                                            DATAPAGE  DB  =DSN8D42A           HASH    =X'01060312'
                                                                    OB  =PROJ               ---------------- BLOCKER IS HOLDER------
                                                                    PAGE=X'000008'          LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3B      CORRID=RUNLOC3B
                                                                                            DURATION=MANUAL     PRIMAUTH=JULIE
                                                                                            STATE   =U
                                                                                            ---------------- WAITER ----------------
                                                                                            LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3A      CORRID=RUNLOC3A
                                                                                            DURATION=MANUAL     PRIMAUTH=BOB
                                                                                            REQUEST =LOCK       WORTH   =   18
                                                                                            STATE   =U
                                                                                            ---------- BLOCKER IS HOLDER --*VICTIM*-
                                                                                            LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3C      CORRID  =RUNLOC3C
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            STATE   =U
                                                                                            ---------------- WAITER ----------------
                                                                                            LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3B      CORRID  =RUNLOC3B
                                                                                            DURATION=MANUAL     PRIMAUTH=JULIE
                                                                                            REQUEST =LOCK       WORTH   =   18
                                                                                            STATE   =U
                                                                                            ---------- BLOCKER IS HOLDER -----------
                                                                                            LUW='BLANK'.EGTVLU2.AAD15D373533
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3A      CORRID  =RUNLOC3A
                                                                                            DURATION=MANUAL     PRIMAUTH=BOB
                                                                                            STATE   =U
                                                                                            ---------------- WAITER -------*VICTIM*-
                                                                                            LUW='BLANK'.EGTVLU2.AB33745CE357
                                                                                            MEMBER  =DB1A       CONNECT =BATCH
                                                                                            PLANNAME=LOC3C      CORRID  =RUNLOC3C
                                                                                            DURATION=MANUAL     PRIMAUTH=SYSADM
                                                                                            REQUEST =LOCK       WORTH   =   18
                                                                                            STATE   =U
```

Figure 44. Deadlock scenario 2: Three transactions and three resources

Related tasks:

Improving concurrency

Monitoring concurrency and locks

Related information:

"Deadlock" on page 189

Monitoring SQL performance

You can monitor the performance of the SQL statements that are run by your application programs.

Related concepts:

Investigating SQL performance by using EXPLAIN

Interpreting data access by using EXPLAIN

Related tasks:

Collecting performance data for SQL statements (Optim Performance Manager)

Importing and viewing Optim Performance Manager SQL performance data in the SQL Outline view (IBM Data Studio)

Generating visual representations of access plans (IBM Data Studio)

Related reference:

EXPLAIN (DB2 SQL)

Monitoring SQL performance with IBM optimization tools

SQL optimization tools, such as IBM Data Studio or IBM Data Server Manager and Optim Query Tuner for DB2 for z/OS, enable you to connect to a DB2 for z/OS subsystem or group member from a workstation computer and collect performance data about SQL workloads and statements that run on that subsystem.

Related tasks:

Collecting performance data for SQL statements (Optim Performance Manager)

Importing and viewing Optim Performance Manager SQL performance data in the SQL Outline view (IBM Data Studio)

Generating visual representations of access plans (IBM Data Studio)

Related reference:

IBM Data Studio product overview (IBM Data Studio)

DB2 Query Workload Tuner for z/OS

Related information:

IBM Data Server Manager

Tuning SQL with Optim Query Tuner, Part 1: Understanding access paths (IBM developerWorks)

DB2-supplied user tables for optimization tools

Query optimization tools, such Optim Query Tuner might create and use one or more instances of certain user tables that are supplied with DB2 on a DB2 subsystem.

Query optimization and analysis tools might create instances of any of the following types of user tables that are supplied with DB2 to enable monitoring, analysis, and tuning functions for queries and query workloads:

- EXPLAIN tables
- Profile tables
- Virtual index table
- Additional user tables that are supplied by the optimization tool.

See the product documentation for detailed information about which types of tables are required to enable each tool.

Related concepts:

Input tables

Related reference:

EXPLAIN tables

Profile tables

Collecting statement-level statistics for SQL statements

You can enable the collection of statement-level statistics for both static and dynamic SQL statements at the DB2 subsystem level.

About this task

Monitor trace class 29 enables you to create READS programs to monitor SQL statements at the DB2 subsystem level through the instrumentation facility interface.

Procedure

To enable the collection of statement level statistics:

Create a program that issues IFI command to enable monitor trace class 29. Your program can monitor static SQL statements, dynamic SQL statements or both:

- For static SQL statements, set up READS calls to monitor IFCID 0401. IFCID 0400 controls the collection of statistics for static SQL statements.
- For dynamic SQL statements, set up READS calls to monitor IFCIDs 0316 and 0317. IFCID 0318 controls the collection of statistics for dynamic SQL statements.

Your program can examine the return areas for statement-level statistics, and can gather EXPLAIN information for statements that have unexpected statistics values.

Related concepts:

Programming for the instrumentation facility interface (IFI)

Chapter 44, "Investigating SQL performance by using EXPLAIN," on page 693

Related tasks:

Controlling the collection of statistics for SQL statements

Monitoring the dynamic statement cache with READS calls

Monitoring static SQL statements with READS calls

Granting authorities for monitoring and tuning SQL statements

You can enable the monitoring and tuning of SQL statements and SQL workloads, without providing additional privileges, such as access to data in the tables that are accessed by the statements.

Before you begin

Your authorization ID or role must have one of the following authorities or privileges:

- ACCESSCTRL (Managing Security)
- SECADM (Managing Security)
- EXPLAIN privilege with WITH GRANT OPTION
- SYSADM (Managing Security) (if the SEPARATE_SECURITY subsystem parameter value is NO)

About this task

You might want to enable a production database administrator, performance analyst, or application developer to complete monitoring and tuning tasks for SQL statements, without giving them access to data in the tables referenced by the statements. The EXPLAIN privilege enables certain EXPLAIN tasks without giving any privileges to access or modify the data. SQLADM authority enables additional monitoring and tuning capabilities, including those enabled by the EXPLAIN privileges, again without providing access to the data in the tables.

Procedure

To enable a user to monitor and tune SQL statements, without access to the data:

- Enable the EXPLAIN privilege, by taking one of the following actions:
  - Issue a GRANT statement:
    ```
    GRANT EXPLAIN TO authorization-ID
    ```
  - Use RACF to permit access to the EXPLAIN system resource.

  The EXPLAIN privilege enables you to:
  - Issue EXPLAIN statements, including PLAN and ALL, without privileges to execute the SQL statements.
  - Run EXPLAIN for dynamic statements that execute under the CURRENT EXPLAIN MODE = EXPLAIN special register
  - Issue PREPARE statements and DESCRIBE TABLE statements, without privileges for the objects.
  - Issue BIND and REBIND commands, and specify the EXPLAIN(ONLY) and SQLERROR(CHECK) options.
- Enable SQLADM authority by issuing a GRANT statement: SQLADM authority enables you to:
  - Issue EXPLAIN statements, including PLAN, ALL, STMTCACHE ALL, STMTID, STMTTOKEN, and MONITORED STMTS, without privileges to execute the SQL statements.
  - Run EXPLAIN for dynamic statements that execute under the CURRENT EXPLAIN MODE = EXPLAIN special register
  - Issue PREPARE statements and DESCRIBE TABLE statements, without privileges for the objects.
  - Issue BIND and REBIND commands, and specify the EXPLAIN(ONLY) and SQLERROR(CHECK) options.
  - Issue START PROFILE, STOP PROFILE, and DISPLAY PROFILE commands.
  - Run the following utilities:
    - DIAGNOSE
    - LISTDEF
    - MODIFY STATISTICS
    - RUNSTATS
    - DSN1SDMP
  - Execute system-defined routines, including stored procedures and functions, and any packages that are defined within the routines.
  - Select data from all catalog tables, and modify data in updatable catalog tables (except for the SYSIBM.SYSAUDITPOLICIES table).

Related concepts:

Investigating SQL performance by using EXPLAIN

Related tasks:

Permitting RACF access (Managing Security)

Related reference:

GRANT (DB2 SQL)

GRANT (system privileges) (DB2 SQL)

Explicit system privileges (Managing Security)

SQLADM (Managing Security)

Facilities and tools for DB2 performance monitoring

EXPLAIN (DB2 SQL)

-START PROFILE (DB2) (DB2 Commands)

Monitoring hash access

You can optimize the performance of hash access in DB2 by monitoring and tuning the storage space that is used by hash access.

About this task

The SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS tables contain indicators that can help you optimize hash access and improve single-row access for queries that use equal predicates.

Procedure

To optimize the use of hash space and improve single-row access on tables that are organized by hash:

- Compare the values for SYSINDEXSPACESTATS.TOTALENTRIES and SYSTABLESPACESTATS.TOTALROWS. If the TOTALENTRIES value is greater than 10 or 15 percent of the value of TOTALROWS, the data might need to be reorganized, or the size of the hash space might need to be increased. However, careful analysis of the row size, page size, and PCTFREE values might be needed to identify when to reorganize a particular hash-organized table. For more information about selecting an appropriate size for the hash space, see Fine-tuning hash space and page size.
- Compare the values for SYSTABLESPACESTATS.DATASIZE and SYSTABLESPACE.HASHSPACE. If DATASIZE is greater than HASHSPACE, increase the size of the hash space. The recommended size for hash spaces depends on the row size and page size. For more information about choosing a size of the hash space, see “Managing space and page size for hash-organized tables” on page 244.
- Monitor SYSTABLESPACESTATS.REORGHASHACCESS to ensure that applications are using hash access paths on tables that are organized by hash. This value represents the number of times that a hash home page is accessed to locate a record for operations such as the following, regardless of whether a qualifying record is found:
  - SELECT
  - FETCH
  - Searched UPDATE
  - Searched DELETE
  - Enforcement of referential integrity constraints

  For partition-by-growth table spaces, the value is only increased for partitions that contain hash home pages. This value is always zero for partitions in a partition-by-growth table space that do not have hash home pages, even though pages in these partitions might be accessed through the hash overflow index.

If the REORGHASHACCESS value does not increase after several queries have accessed the table, the hash access path is not being used regularly. Consider removing hash organization from such tables to conserve storage space. The value of REORGHASHACCESS resets to zero after the REORG TABLESPACE or LOAD REPLACE utility is run.

- Monitor SYSTABLESPACESTATS.HASHLASTUSED to ensure that applications have used hash access paths on the table recently. If the DB2 has not recently used a hash access path on the table, consider removing hash organization from the table to conserve storage space.

Related tasks:

Organizing tables by hash for fast access to individual rows

Altering the size of your hash spaces (DB2 Administration Guide)

Related reference:

SYSIBM.SYSINDEXSPACESTATS table (DB2 SQL)

SYSIBM.SYSTABLESPACESTATS table (DB2 SQL)

REORG TABLESPACE (DB2 Utilities)

Related information:

Monitoring the performance of hash access tables (DB2 10 for z/OS Technical Overview)

Monitoring the performance of hash access tables (DB2 10 for z/OS Performance Topics)

Gathering information about SQL statements for IBM Software Support

You can use the DSNADMSB program to gather information about poorly performing SQL statements when you contact IBM Software Support.

About this task

The DSNADMSB program collects environment information about SQL statements, such as the following types of information:

- Objects such as tables, indexes and views that are referenced by the statements
- Statistics
- EXPLAIN table information
- DB2 module details
- Subsystem parameter settings

IBM Software Support can use this information to re-create and analyze the performance problems.

Related reference:

DSNADMSB (DB2 Utilities)

Monitoring parallel operations

You can monitor the use of parallelism by DB2.

About this task

The number of parallel tasks that DB2 uses to access data is determined at bind time, and is adjusted again when the query is executed.

Bind time

At bind time, DB2 collects partition statistics from the catalog, estimates the processor cycles for the costs of processing the partitions, and determines the optimal number of parallel tasks to achieve minimum elapsed time.

When a planned degree exceeds the number of online CPs, the query might not be completely processor-bound. Instead it might be approaching the number of partitions because it is I/O-bound. In general, the more I/O-bound a query is, the closer the degree of parallelism is to the number of partitions.

In general, the more processor-bound a query is, the more degree of parallelism is related to the to the number of online CPs. However, the degree of parallelism can exceed the number of CPs. The default degree of parallelism is twice the number of CPs. A degree of parallelism that is greater than the number of CPs is beneficial in cases where works is distributed unevenly among parallel child tasks.

To help DB2 determine the optimal degree of parallelism, use the RUNSTATS utility to keep your statistics current.

You can find the expected degree of parallelism in the ACCESS_DEGREE and JOIN_DEGREE columns of the PLAN_TABLE.

Execution time

For each parallel group, parallelism can execute at a reduced degree or degrade to sequential operations for the following reasons:

- Amount of virtual buffer pool space available
- Host variable values
- Availability of the hardware sort assist facility
- Ambiguous cursors
- A change in the number or configuration of online processors
- The join technique that DB2 uses (I/O parallelism is not supported when DB2 uses the star join technique)

At execution time, a plan using Sysplex query parallelism can use CP parallelism. All parallelism modes can degenerate to a sequential plan. No other changes are possible.

Procedure

To monitor parallel operations, use one of the following approaches:

1. Use the DISPLAY BUFFERPOOL report to see how well the buffer pool is able to satisfy parallel operations.

```
DSNB440I = PARALLEL ACTIVITY -
           PARALLEL REQUEST =      282  DEGRADED PARALLEL=       5
```

   The PARALLEL REQUEST field in this example shows that DB2 was negotiating buffer pool resource for 282 parallel groups. Of those 282 groups, only 5 were degraded because of a lack of buffer pool resource. A large number in the DEGRADED PARALLEL field could indicate that your subsystem does not have enough buffers that can be used for parallel processing.

2. Use the DISPLAY THREAD command. The status field contains PT for parallel tasks. All parallel tasks are displayed immediately after their corresponding originating thread.
3. Use DB2trace:
   - The statistics trace indicates The statistics trace indicates when parallel groups do not run with the planned degree of parallelism or run sequentially. Either situation might indicate that some queries do not achieve the best possible response times.
   - PSPI You can use the accounting trace to ensure that your parallel queries are meeting their response time goals. DB2 rolls task accounting into an accounting record for the originating task. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS also summarizes all accounting records generated for a parallel query and presents them as one logical accounting record.

     Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS presents the times for the originating tasks separately from the accumulated times for all the parallel tasks.

     As shown in the following accounting trace record excerpt, `CP CPU TIME-AGENT` is the time for the originating tasks, while `CP CPU TIME-PAR.TASKS` is the accumulated processing time for the parallel tasks.

```
TIMES/EVENTS  APPL (CLASS 1)   DB2 (CLASS 2)    CLASS 3 SUSP.   ELAPSED TIME
------------  --------------  --------------    --------------  ------------
ELAPSED TIME       32.578741       32.312218    LOCK/LATCH         25.461371
 NONNESTED         28.820003       30.225885    SYNCHRON. I/O       0.142382
 STORED PROC        3.758738        2.086333     DATABASE I/O       0.116320
 UDF                0.000000        0.000000     LOG WRTE I/O       0.026062
 TRIGGER            0.000000        0.000000    OTHER READ I/O   3:00.404769
                                                OTHER WRTE I/O      0.000000
CPU CP TIME      1:29.695300     1:29.644026    SER.TASK SWTCH      0.000000
 AGENT              0.225153        0.178128     UPDATE COMMIT      0.000000
  NONNESTED         0.132351        0.088834     OPEN/CLOSE         0.000000
  STORED PRC        0.092802        0.089294     SYSLGRNG REC       0.000000
  UDF               0.000000        0.000000     EXT/DEL/DEF        0.000000
  TRIGGER           0.000000        0.000000     OTHER SERVICE      0.000000
 PAR.TASKS       1:29.470147     1:29.465898    ARC.LOG(QUIES)      0.000000

:
...            QUERY PARALLEL.     TOTAL
               ---------------  --------
               MAXIMUM MEMBERS         1
               MAXIMUM DEGREE         10
               GROUPS EXECUTED         1
                RAN AS PLANNED         1
                RAN REDUCED            0
                ONE  COOR=N            0
                ONE  ISOLAT            0
                ONE  DCL TTABLE        0
                SEQ - CURSOR           0
                SEQ - NO ESA           0
                SEQ - NO BUF           0
                SEQ - ENCL.SER.        0
               MEMB SKIPPED(%)         0
               DISABLED BY RLF        NO
               REFORM PARAL-CONFIG     0
               REFORM PARAL-NO BUF     0
```

Figure 45. A partial sample that shows parallelism fields in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting trace report

  As the report shows, the values for `CPU TIME` and `I/O WAIT TIME` are larger than the elapsed time. The processor and suspension time can be greater than the elapsed time because these two times are accumulated from multiple parallel tasks. The elapsed time would be less than the processor and suspension time if these two times are accumulated sequentially.

  If you have baseline accounting data for the same thread run without parallelism, the elapsed times and processor times must not be larger when that query is run in parallel. If they are larger, or if response time is poor, you need to examine the accounting data for the individual tasks. Use the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Record Trace for the IFCID 0003 records of the thread you want to examine. Use the performance trace if you need more information to determine the cause of the response time problem.

- If you discover a potential problem with a parallel query, you can use the performance trace to do further analysis. You can refer to field QW0221AD in IFCID 0221, as mapped by macro DSNDQW03. The 0221 record also gives you information about the key ranges used to partition the data.

  IFCID 0222 contains the elapsed time information for each parallel task and each parallel group in each SQL query. Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS presents this information in its SQL Activity trace.

  If your queries are running sequentially or at a reduced degree because of a lack of buffer pool resources, the QW0221C field of IFCID 0221 indicates which buffer pool is constrained.

PSPI

Related reference:

-DISPLAY BUFFERPOOL (DB2) (DB2 Commands)

-DISPLAY THREAD (DB2) (DB2 Commands)

PLAN_TABLE

Accounting Long Report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

SQL Activity Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Monitoring DB2 in distributed environments

You can use DISPLAY commands and the trace facility to obtain information. DB2 can also return server-elapsed time to certain types of client applications.

Introductory concepts

Distributed data (Introduction to DB2 for z/OS)

Distributed data access (Introduction to DB2 for z/OS)

Effects of distributed data on planning (Introduction to DB2 for z/OS)

The DB2 DISPLAY commands provide information about the status of threads, databases, tracing, allied subsystems, and applications. The following DISPLAY commands are particularly helpful for monitoring DB2 in distributed environments:

- DISPLAY DATABASE
- DISPLAY DDF DETAIL

- DISPLAY LOCATION
- DISPLAY THREAD
- DISPLAY TRACE

Related concepts:

Monitoring tools for distributed environments

Related tasks:

“Monitoring threads and connections by using profiles” on page 108

Limiting resources for statements from remote locations

Monitoring threads (DB2 Administration Guide)

Displaying information about connections with other locations (DB2 Administration Guide)

Displaying information about DDF work (DB2 Administration Guide)

Tracing distributed events

A number of IFCIDs, including IFCID 0001 (statistics) and IFCID 0003 (accounting), record distributed data and events.

PSPI

If your applications update data at other sites, turn on the statistics class 4 trace and always keep it active. This statistics trace covers error situations surrounding in doubt threads; it provides a history of events that might impact data availability and data consistency.

DB2 accounting records are created separately at the requester and each server. Events are recorded in the accounting record at the location where they occur. When a thread becomes active, the accounting fields are reset. Later, when the thread becomes inactive or is terminated, the accounting record is created.

The following figure shows the relationship of the accounting class 1 and 2 times and the requester and server accounting records.

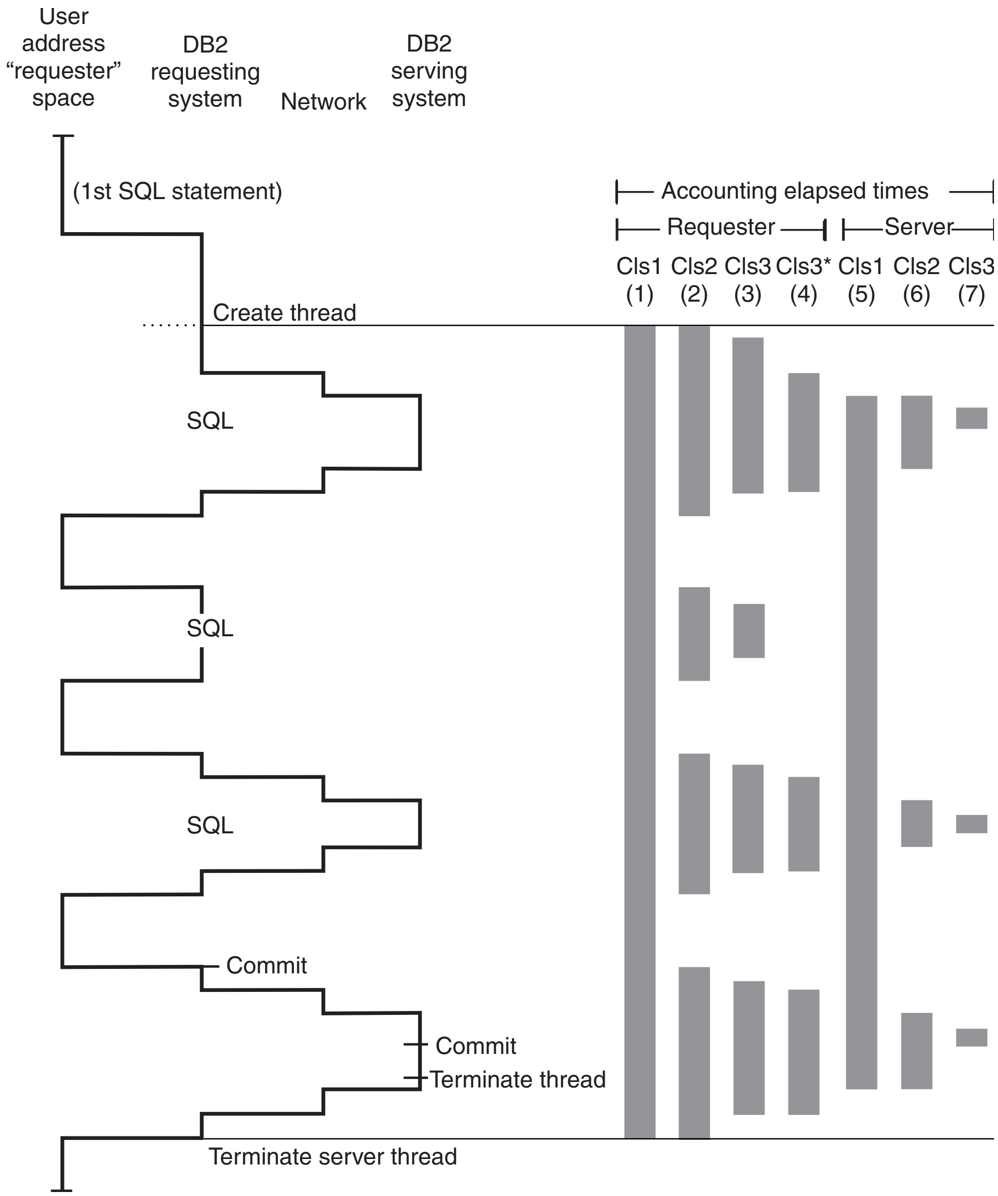


Figure 46. Elapsed times in a DDF environment as reported by Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS. These times are valid for access that uses DRDA(except as noted).

This figure is a simplified picture of the processes that go on in the serving system. It does not show block fetch statements and is only applicable to a single row retrieval.

The various elapsed times referred to in the header are:

- (1) - Requester Cls1

  This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting long trace for the requesting DB2 subsystem. It represents the elapsed time from the creation of the allied distributed thread until the termination of the allied distributed thread.
- (2) - Requester Cls2

  This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting long trace for the requesting DB2 subsystem. It represents the elapsed time from when the application passed the SQL statements to the local DB2 system until return. This is considered "In DB2" time.

- (3) - Requester Cls3

  This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting long trace for the requesting DB2 system. It represents the amount of time the requesting DB2 system spent suspended waiting for locks or I/O.
- (4) - Requester Cls3* (Requester wait time for activities not in DB2)

  This time is reported in the SERVICE TASK SWITCH, OTHER SERVICE field of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report for the requesting DB2 subsystem. It is typically time spent waiting for the network and server to process the request.
- (5) - Server Cls1

  This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting long trace for the serving DB2 subsystem. It represents the elapsed time from the creation of the database access thread until the termination of the database access thread.
- (6) - Server Cls2

  This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting long trace of the serving DB2 subsystem. It represents the elapsed time to process the SQL statements and the commit at the server.
- (7) - Server Cls3

  This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting long trace for the serving DB2 subsystem. It represents the amount of time the serving DB2 system spent suspended waiting for locks or I/O.

The Class 2 processing time (the TCB time) at the requester does not include processing time at the server. To determine the total Class 2 processing time, add the Class 2 time at the requester to the Class 2 time at the server.

Likewise, add the getpage counts, prefetch counts, locking counts, and I/O counts of the requester to the equivalent counts at the server. For DRDA, SQL activity is counted only at the server.

PSPI

Reporting server-elapsed time

Client applications that access DB2 data using DRDA access can request that DB2 return the server-elapsed time.

Server-elapsed time allows remote clients to determine the actual amount of time it takes for DB2 to parse a remote request, process any SQL statements required to satisfy the request, and generate the reply. Because server-elapsed time does not include any of the network time used to receive the request or send the reply, client applications can use the information to quickly isolate poor response times to the network or to the DB2 server without you having to perform traces on the server. When the System Monitor statement switch has been turned on, DB2 returns server-elapsed time information as a new element through the regular Snapshot Monitoring APIs.

Monitoring distributed processing with RMF

If you use RMF to monitor DDF work, understand how DDF is using the enclave SRBs.

The information that is reported using RMF or an equivalent product in the SMF 72 records are the portions of the client's request that are covered by individual enclaves. The way DDF uses enclaves relates directly to whether the DDF thread can become inactive.

Related information:

Effective zSeries Performance Monitoring Using Resource Measurement Facility (IBM Redbooks)

Duration of an enclave

If a thread is always active, the duration of the thread is the duration of the enclave. Otherwise, certain conditions control the duration of an enclave.

If the thread can be pooled, the following conditions determine the duration of an enclave:

- If the associated package is bound with KEEPDYNAMIC(NO), or there are no open held cursors, or there are active declared temporary tables, the duration of the enclave is the period during which the thread is active.
- If the associated package is bound with KEEPDYNAMIC(YES), and no held cursors or active declared temporary tables exist, and only KEEPDYNAMIC(YES) keeps the thread from being pooled, the duration of the enclave is the period from the beginning to the end of the transaction.

While a thread is pooled, such as during think time, it is not using an enclave. Therefore, SMF 72 record does not report inactive periods.

ACTIVE MODE threads are treated as a single enclave from the time it is created until the time it is terminated. This means that the entire life of the database access thread is reported in the SMF 72 record, regardless of whether SQL work is actually being processed. Figure 47 on page 653 contrasts the two types of threads.

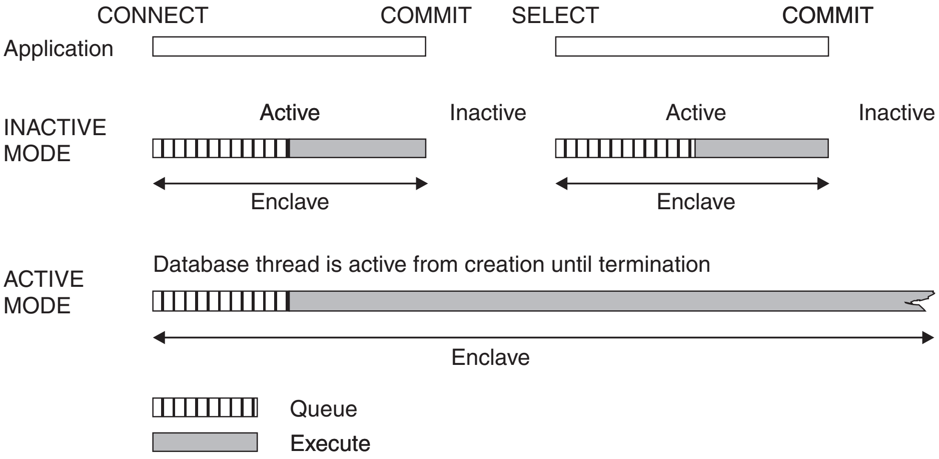


Figure 47. Contrasting ACTIVE MODE threads and POOLED MODE threads

Queue time: Note that the information that is reported back to RMF includes queue time. This particular queue time includes waiting for a new or existing thread to become available.

RMF records for enclaves

You can find the total enclave usage in the RMF record, but you must use DB2 accounting traces to see resource consumption for a particular enclave.

The two most frequently used SMF records are types 30 and 72. The type 30 record contains resource consumption at the address space level.

Each enclave contributes its data to one type 72 record for the service class and to zero or one (0 or 1) type 72 records for the report class. You can use WLM classification rules to separate different enclaves into different service or report classes. Separating the enclaves in this way enables you to understand the DDF work better.

| You can use the RMF Monitor I workload activity report to look at the related
| WLM service class and reporting class information.

Related concepts:

Accounting trace

Related reference:

➪ z/OS RMF Report Analysis

➪ z/OS MVS System Management Facilities (SMF)

Monitoring use of IBM specialty engines

You can use various facilities to monitor how DB2 uses IBM Z Integrated Information Processor (zIIP) and IBM Z Application Assist Processor (zAAP).

Procedure

To monitor IBM specialty engine usage:

- Use DB2 trace.

  The DB2 accounting trace records provide information related to application programs including processor resources consumed by the application.

Accumulated specialty engine CPU time is not accumulated in the existing class 1, class 2, and class 7 accounting fields. New accounting fields are defined to let you know how much time is spent on specialty engines, as well as how much specialty engine eligible work overflowed to standard processors.

- Use RMF.

  The Resource Measurement Facility (RMF) provides information on specialty engine usage to help you identify when to consider purchasing a specialty engine or adding more specialty engines. Also, SMF Type 72 records contain information on specialty engine usage and fields in SMF Type 30 records let you know how much time is spent on specialty engines, as well as how much time was spent executing specialty engine eligible work on standard processors.

Related concepts:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related reference:

z/OS RMF User's Guide

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

IBM Z Integrated Information Processor (zIIP) specialty engines

Portions of certain processing are eligible for dispatching, as implemented by IBM Z Integrated Information Processor (zIIP) specialty engines. The amount of general-purpose processor savings varies based on the amount of workload executed by the zIIP specialty engine, among other factors.

Authorized uses for IBM Z Integrated Information Processor (zIIP)

Some or all of the following processes are authorized [1] and eligible for dispatching to IBM Z Integrated Information Processor (zIIP):

SQL request workloads that use DRDA to access DB2 for z/OS over TCP/IP connections [2]

Up to 60% of the DB2 for z/OS instructions executing such SQL requests, when running in Enclave SRB Mode and accessing DB2 for z/OS.

Parallel query child processes [2]

After reaching a CPU usage threshold, up to 80% of the processing of long-running parallel queries for DB2 for z/OS. Long running parallel queries are those which the DB2 for z/OS Query Optimizer determines should be run in parallel and whose execution exceeds an identified period of time as established by DB2 for z/OS (the "CPU usage threshold.") The CPU usage threshold is defined by IBM uniquely for each IBM Z Machine type.

Utility processes

Including the following processes:

- Up to 100% of the portion of LOAD, REORG, and REBUILD INDEX utility function that is used to maintain index structures.
- The DB2 RUNSTATS utility instructions for RUNSTATS options. (With the exception of distribution statistics and inline statistics.)

XML processing

Including the following processes:

- Up to 100% of XML schema validation and non-validation parsing.
- Up to 100% of the deletion of unneeded versions of XML documents.

DB2 buffer pools
Up to 100% of the DB2 instructions that execute buffer pool prefetch and deferred write processing.

Notes:

1. This information provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (for example, zIIP, zAAP, and IFLs) ("SEs"). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the "Authorized Use Table for IBM Machines" provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html ("AUT").

   No other workload processing is authorized for execution on an SE. IBM offers SE at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.
2. When executing on a System 9, System 10, or IBM Z Enterprise Servers (z196 or z114) or subsequent machines.
3. z/OS 1.10 is required for these instructions to be eligible to run on zIIP specialty engines.

Related reference:

IBM Z Integrated Information Processor (zIIP)

IBM Z Integrated Information Processor (zIIP) Workloads

IBM IBM Z Application Assist Processor (zAAP)

The IBM Z Application Assist Processor (zAAP) is a specialty engine that provides an execution environment for certain service-oriented architecture (SOA) technologies, such as Java and XML.

TheIBM Z Application Assist Processor (zAAP) is a specialty engine that can run eligible Java and XML processing for database workloads. zAAP specialty engines are designed to free general computing capacity and lower software costs for certain web-based and SOA-based DB2 workloads, such as Java and XML.

The amount of general-purpose processor savings varies based on the amount of workload that the zAAP specialty engines run, among other factors.

Java

zAAP specialty engines might help increase general-purpose processor productivity and might lower the overall cost of computing for Java technology-based applications that run on z/OS. zAAP specialty engines are designed to operate asynchronously with the general processors to run Java programming under control of the IBM Java virtual machine (JVM). They can help reduce the demands and capacity requirements on general-purpose processors, which might then be available for reallocation to other mainframe workloads.

XML

zAAP specialty engines can help simplify and reduce server infrastructures by enabling the development of XML-based applications that are integrated with existing data stores. z/OS XML System Services (z/OS XML), is a system-level XML parser that is integrated with the base z/OS operating system and designed to deliver an optimized set of services for parsing XML documents. DB2 uses z/OS XML System Services for non-validation and validation parsing for a portion of SQL/XML processing. For applications that run locally on z/OS and insert, update, or load XML data into tables, DB2 starts z/OS XML, which runs on zAAP specialty engines. The remaining DB2 XML processing is run on general-purpose processors.

Related concepts:

pureXML (Introduction to DB2 for z/OS)

Java application development for IBM data servers (DB2 Application Programming for Java)

Related reference:

IBM Z Application Assist Processor (zAAP)

Related information:

Service Oriented Architecture (SOA)

Checking for invalid packages

You can create a query to check for plans and packages that have become invalid or inoperative.

Procedure

To check for invalid packages:

GUPI

Issue the following SQL statement:

```
SELECT COLLID, NAME, VERSION, VALIDATE, ISOLATION, VALID, OPERATIVE
  FROM SYSIBM.SYSPACKAGE
  WHERE VALIDATE = 'R' OR ISOLATION = 'R'
    OR VALID = 'N' OR OPERATIVE = 'N';
```

GUPI

These statements identify packages that meet the following criteria:

- Might redo validity checks at run time; if an invalid object or missing authority is found, DB2 issues a warning and checks again for the object or authorization at run time.
- Use repeatable read isolation.
- Are invalid (must be rebound before use), for example, deleting an index or revoking authority can render a package invalid.
- Are inoperative (require an explicit BIND or REBIND before use). A package can be marked inoperative after an unsuccessful REBIND operation.

Related concepts:

Changes that invalidate packages (DB2 Application programming and SQL)

Automatic rebinds (DB2 Application programming and SQL)

The ISOLATION (RR) option

Related tasks:

Rebinding a package (DB2 Application programming and SQL)

Related reference:

Invalid and inoperative packages (Managing Security)

SYSIBM.SYSPACKAGE table (DB2 SQL)

Related information:

00E30305 (DB2 Codes)

Using profiles to monitor and optimize DB2 for z/OS subsystems

Profile tables enable you to monitor the use of system resources, control performance-related subsystem parameters in particular contexts on your DB2 subsystem. Each monitored context is defined by a set of criteria called a profile.

About this task

A *profile* is a set of criteria that identifies a particular context on a DB2 subsystem. Examples include threads, connections, or SQL statements that have particular attributes.

Related tasks:

Monitoring threads and connections by using profiles

Modeling a production environment on a test subsystem

Optimizing subsystem parameters for SQL statements by using profiles

Related reference:

Profile tables

Profiles for monitoring and controlling DB2 for z/OS subsystems

A *profile* is a set of criteria that identifies a particular context on a DB2 subsystem. Examples include threads, connections, or SQL statements that have particular attributes.

Overview of uses for profiles

You can create profiles to define filtering scopes for processes within DB2. The filtering for each profile is controlled by the column values in the DSN_PROFILE_TABLE. You can also specify actions for DB2 to take when a process, such as a SQL statement, thread, or connection meets the criteria of the profile. The actions are specified by the column values in the DSN_PROFILE_ATTRIBUTES table. The actions that you can specify include:

- Monitoring threads and connections
- Setting or disabling certain subsystem parameters for particular SQL statements
- Specifying subsystem properties when modeling a production environment on a test system
- Setting thresholds for query acceleration
- Evaluating a dynamic SQL query for acceleration

Many of these actions are not closely related. The rules for how to define the scope of the profile depend on the action that the profile defines. Certain combinations of filtering columns are either required or accepted only for certain KEYWORDS values. The following table summarized these relationships. For detailed information about the accepted filtering columns and KEYWORDS values, see the specific information for each action.

Table 118. Summary of uses for profiles, applicable filtering columns, and applicable KEYWORDS values

Profile action	Applicable filtering columns	Applicable KEYWORDS values
Monitoring the use of system resources by remote applications, including remote threads and connections [1]	Any of the following combinations with particular rules for certain KEYWORDS values: • LOCATION only • PRDID only • AUTHID, ROLE, or both. • COLLID, PKGNAME, or both • One of CLIENT_APPLNAME, CLIENT_USERID, CLIENT_WORKSTNNAME	• MONITOR CONNECTIONS • MONITOR THREADS • MONITOR IDLE THREADS
Setting or disabling optimization parameters for SQL statements	All of the following columns: • PLANNAME set to '*' • COLLID • PKGNAME	• MIN STAR JOIN TABLES • NPAGES THRESHOLD • STAR JOIN • IO WEIGHTING
Modeling a production system on a test system	None. Profiles for this purpose have a global scope on the test subsystem.	• BP*name* • MAX_RIDBLOCKS • SORT_POOL_SIZE
Setting thresholds for query acceleration	Contact IBM Support for the accelerator product.	• ACCEL_TABLE_THRESHOLD • ACCEL_RESULTSIZE_THRESHOLD • ACCEL_TOTALCOST_THRESHOLD
Evaluating a dynamic SQL query for acceleration	Supported by profiles with a global scope and those with the following combinations: • None (as a global scope) • AUTHID and LOCATION • PLANNAME, COLLID, and PKGNAME	ACCEL_NAME_EXPLAIN

Notes:

1. Profiles of this type apply only to remote applications.

The scope of a profile

The scope of a profile, the defined context in which DB2 applies the actions specified by the profile, are stored as rows in the SYSIBM.DSN_PROFILE_TABLE table. Rows in the SYSIBM.DSN_PROFILE_ATTRIBUTES table control the actions that DB2 applies when a process meets the criteria defined by the profile. The values of the PROFILEID columns of each table associate each profile with the corresponding actions for that profile. The PROFILE_ENABLED column indicates whether DB2 activates the profile when you start monitoring.

How you define a profile depends on the context that you want to define and the actions that you want DB2 to perform. A valid row in

SYSIBM.DSN_PROFILE_TABLE always contains null values in some columns. Which of the following columns that you define depend on the purpose of the profile:

- AUTHID
- COLLID
- LOCATION
- PKGNAME
- PLANNAME
- PRDID
- ROLE
- CLIENT_APPLNAME
- CLIENT_USERID
- CLIENT_WRKSTNNAME

For profiles that specify values for optimizing subsystem parameters, the following table shows combinations of criteria that create valid profiles. Other combinations are not valid.

Table 119. Categories and columns used to specify valid profiles that specify subsystem parameters

Filtering category	Columns to specify
Collection identifier and package name	Specify all of the following columns: • PLANNAME (specify only '*') • COLLID • PKGNAME

For profiles that define contexts for the use of system resources by threads and connections, the following combinations of criteria create valid profiles. Other combinations are not valid.

Table 120. Categories and columns used to specify valid profiles for monitoring system resources

Filtering category	Columns to specify
Client IP address or domain name	Specify only the LOCATION column. The value can be an IP address or domain name. This category is the only accepted filtering criteria for profiles that specify the MONITOR CONNECTIONS.
Client product identifier	Specify only the PRDID column.
Role or authorization identifier	Specify one or all of the following columns: • ROLE • AUTHID Profiles that specify both ROLE and AUTHID take precedence over profiles that specify only one value. Profiles that specify only ROLE take precedence over profiles that specify only AUTHID

Table 120. Categories and columns used to specify valid profiles for monitoring system resources (continued)

Filtering category	Columns to specify
Collection identifier or package name	Specify only one or all of the following columns: • COLLID • PKGNAME Profiles that specify both COLLID and PKGNAME take precedence over profiles that specify only one value. Profiles that specify only COLLID take precedence over profiles that specify only PKGNAME
Location name, or location alias	Specify only the location name or location alias in LOCATION column. This category applies only to profiles that specify MONITOR THREADS and MONITOR IDLE THREADS.
Client application name, user ID, or workstation ID.	Specify only one of the following columns: • CLIENT_APPLNAME • CLIENT_USERID • CLIENT_WRKSTNNAME

Related tasks:

Creating profiles

Monitoring threads and connections by using profiles

Modeling a production environment on a test subsystem

Optimizing subsystem parameters for SQL statements by using profiles

Related reference:

Profile tables

Creating profiles

You can create profiles to define filtering scopes for processes within DB2. The filtering for each profile is controlled by the column values in the DSN_PROFILE_TABLE. You can also specify actions for DB2 to take when a process, such as a SQL statement, thread, or connection meets the criteria of the profile. The actions are specified by the column values in the DSN_PROFILE_ATTRIBUTES table.

Before you begin

Before you can create profiles, you must create a set of profile tables on the DB2 subsystem.

About this task

A *profile* is a set of criteria that identifies a particular context on a DB2 subsystem. Examples include threads, connections, or SQL statements that have particular attributes.

Procedure

To create a profile:

1. Insert rows into SYSIBM.DSN_PROFILE_TABLE with appropriate set of values for the type of profile that you want to create.
2. Insert rows into the SYSIBM.DSN_PROFILE_ATTRIBUTES table to indicate the action that DB2 takes when the criteria that are specified in the profile table are met. DB2 uses the PROFILEID columns to associate rows in the SYSIBM.DSN_PROFILE_TABLE and SYSIBM.DSN_PROFILE_ATTRIBUTES profile tables.

What to do next

After you create a profile, you can enable and disable profiles and issue START PROFILE and STOP PROFILE commands to control which profiles are active.

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

Monitoring threads and connections by using profiles

Modeling a production environment on a test subsystem

Optimizing subsystem parameters for SQL statements by using profiles

Related reference:

SYSIBM.DSN_PROFILE_TABLE

Profile tables

-START PROFILE (DB2) (DB2 Commands)

-STOP PROFILE (DB2) (DB2 Commands)

Starting and stopping profiles

You must enable and start profiles before DB2 can use the information in the profile tables. When you apply changes to an existing started profile, you must stop the profile and start it again before the changes are applied.

Before you begin

Before you can start profiles the following prerequisites must be met:

- A set of profiles exist on the DB2 subsystem.
- The SYSIBM.DSN_PROFILE_TABLE and SYSIBM.DSN_PROFILE_ATTRIBUTES contain rows of data that define valid profiles and the action that DB2 takes when a context meets the definition of a profile.

About this task

A *profile* is a set of criteria that identifies a particular context on a DB2 subsystem. Examples include threads, connections, or SQL statements that have particular attributes.

Procedure

To start or stop profiles, use one of the following approaches:

- Issue a START PROFILE command. DB2 activates the functions specified in the profile tables for every valid row of the SYSIBM.DSN_PROFILE_TABLE table that contains PROFILE_ENABLED='Y'. Profiles in rows that contain PROFILE_ENABLED='N' are not started.

- Issue a STOP PROFILE command. DB2 disables all profile functions.

What to do next

If you modify existing started profiles, you must stop and start profiles again to apply the changes. For profiles that affect access path selection, you must invalidate the statements in the dynamic statement cache before changes to profile attribute values are applied for those statements.

Related tasks:

Invalidating statements in the dynamic statement cache

Related reference:

-START PROFILE (DB2) (DB2 Commands)

-STOP PROFILE (DB2) (DB2 Commands)

Modifying existing profiles

You must take certain actions to apply changes to profiles.

Procedure

To modify existing profiles and apply the changes:

1. Insert, update, or delete from the profile tables to define the changes.
2. Issue a STOP PROFILE command.
3. For changes to profiles for optimization parameters, production system modeling, or query acceleration thresholds, invalidate the statements in the statement cache to apply the access path changes. For information about completing this action, see “Invalidating statements in the dynamic statement cache” on page 412.
4. Issue a START PROFILE command.

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related reference:

Profile tables

-START PROFILE (DB2) (DB2 Commands)

-STOP PROFILE (DB2) (DB2 Commands)

Chapter 42. Investigating DB2 performance problems

Many different logs, traces, and reports are available for analyzing DB2 performance problems, and you might find it hard to know where to look first. By taking a systematic approach, you can focus your investigation and determine the nature of the problem more quickly.

Before you begin

Depending on the symptoms, the first step of investigating a performance problem might be to look at the overall system to determine whether the problem might be outside of DB2:

- Analyze why application processes are progressing slowly, or why a given resource is being heavily used. The best tools for that kind of investigation include:
  - The resource measurement facility (RMF) of z/OS
  - IBM Tivoli Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS XE for DB2 Performance Expert on z/OS.
- If the application processes are running on distributed platforms, analyze the performance on those platforms and the network that connects them.

After you determine that the problem is inside DB2, you can begin detailed analysis of performance data that you captured during monitoring.

About this task

If your initial analysis suggests that the performance problem is within DB2, the problem might be poor response time, an unexpected and unexplained high use of resources, or locking conflicts.

Procedure

To investigate DB2 performance problems.

1. Activate the necessary trace classes. The following trace classes are recommended, as needed:
   - Accounting trace classes:
     - 1, 2, and 3 for plans.
     - 7, and 8 for packages, when needed.
     - 10 for package details, when needed. This class is resource intensive.
   - Statistics trace classes:
     - 1, 3, and 4 for non-data sharing environments.
     - 5 for data sharing environments.
     - 7 for distributed location statistics
     - 8 for buffer pool data set statistics (IFCID 0199).
2. Use the trace data to further narrow your performance investigation.
   - Use DB2 statistics trace data to analyze performance problems at the subsystem level.
   - Use DB2 accounting trace data to analyze application performance problems at the thread level.

Related concepts:

Response times

Suspensions and wait time

Statistics trace

Accounting trace

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related reference:

z/OS RMF User's Guide

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

Investigating CPU performance regression

When you encounter a possible CPU performance regression during migration to a new release, the first challenges are to verify that a CPU regression actually occurred, and to identify the specific connection types, plans, and packages that are involved.

About this task

To do so, you need to find valid comparison points in real production environments, both before and after migration to the new release. The best approach is to exclude batch processing because it is highly variable based on the based on the operational business calendar.

You can then compare the performance data for the period on the previous DB2 release to the corresponding period the new DB2 release. As a starting point, you can use a combination of statistics trace data, accounting trace data, and workload indicators to ensure that you have a valid comparison, and to identify the nature of the problem.

Procedure

To determine whether a CPU regression occurred at migration:

1. Find an interval of several days that has a comparable SQL profile across DB2 releases. For a valid comparison, you need a corresponding interval that has a similar number of total SQL requests, and a similar distribution across the different types of SQL statements, such as SELECT, INSERT, UPDATE, DELETE, and so on.

   If you find that the SQL profile is changed significantly, the application workload has changed and a valid comparison for CPU regression is not possible.
2. Compare performance data for the identified period in the previous release to the corresponding period in the new release. You can use a combination of the statistics and accounting traces to check that you have the same pattern across DB2 releases.
   a. In the statistics trace data, start by comparing the CPU times for the following contexts:
      - Task control blocks (TCB) and service request blocks (SRB) for the *ssnm*MSTR address space.

   - Task control blocks (TCB), service request blocks (SRB), and specialty engine service request blocks for the *ssnm*DBM1 address space. The split between central processor and specialty engine time for the *ssnm*DBM1 address space is likely to be different in Version 10 when compared to Version 8 or Version 9.
   - Task control blocks (TCB) and service request blocks (SRB) for the IRLM address space.

   b. In the accounting trace data, compare the class 2 CPU times for each connection type, for central processors and for specialty engines, and check for the numbers of SQL requests, including the following workload indicators:
      - The numbers of SQL statements for data manipulation, by type of statement (SELECT, INSERT, UPDATE, FETCH, and so on.)
      - The numbers of commit operations, roll back operations, getpage operations, and buffer pool updates.
      - The amount of read and write activity in terms of I/O operations and pages.

   c. Combine the statistics trace data and accounting trace data:
      1) Normalize the values by dividing the CPU time values by the number of commit and roll back operations. The resulting values represent the average “CPU milliseconds per transaction.”
      2) Stack the various components of CPU resource consumption and graph them.

      For example:

```
MSTR TCB cpu-time / (commits + rollbacks)
MSTR SRB cpu-time / (commits + rollbacks)
DBM1 TCB cpu-time / (commits + rollbacks)
DBM1 SRB cpu-time / (commits + rollbacks)
DBM1 IIP SRB cpu-time / (commits + rollbacks)
IRLM TCB cpu-time / (commits + rollbacks)
IRLM SRB cpu-time / (commits + rollbacks)
Average Class 2 CP CPU * occurrences / (commits + rollbacks)
Average Class 2 SE CPU * occurrences / (commits + rollbacks)
```

3. Compare the number of getpage operations for the corresponding intervals. If you find significant changes to the numbers of getpage operations across releases (for comparable application workloads), access path changes are the most likely cause of the CPU regression.

What to do next

You can analyze the details of the accounting data to locate the particular plan or package that is the source of the problem.

Related concepts:

Steps to investigate CPU performance on release migration (DB2 for z/OS Best Practices)

Statistics trace

Accounting trace

Related tasks:

Narrowing your application performance investigation

Investigating thread-level application performance

Investigating access path problems

Monitoring I/O activity of data sets

Grouping data using the connection type identifier (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related reference:

Statistics Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Major contributors to CPU time

You can use CPU time values in the statistics trace report to identify specific processes for further investigation when you analyze the performance of your DB2 subsystem.

The following table shows process that are likely candidates for further investigation based on the particular CPU time values that you find in statistics trace report.

Table 121. Major contributors to CPU time in a DB2 subsystem

	Applications	**Database services (*ssnm*DBM1) address space**	**System services (*ssnm*MSTR) space**	**Internal resources lock manager (IRLM)**
TCB	• SQL processing • Synchronous I/O • Lock requests • Logical locking • Buffer updates • Group buffer pool reads[1] • Global lock requests[1]	• Opening and closing of data sets • DBM1 address space full • System contraction • Preformat • Extend	• Archiving • BSDS processing	• Error checking • Management
SRB	The same contributors as for TCB, but in preemptable enclave SRB mode for DDF applications. The values are reported in the accounting TCB Class 1 and Class 2 CPU instrumentation.	• Deferred writes • Prefetch reads • Parallel child tasks • Castouts[1] • Asynchronous group buffer pool writes[1] • P-lock negotiation[1] • Notify exit[1] • Page set close or pseudo-close to convert to non-GBP dependent[1] • Group buffer pool checkpoints[1]	• Physical log writes • Thread deallocation • Update commits (including unlocking of page P-locks[1]) • Backouts • Checkpoints	• Local IRLM latch contention • IRLM and XES global contention[1] • Asynchronous XES contention[1] • P-lock negotiation[1] • Deadlock detection

notes:

1. Applies to data sharing environments only.

Related concepts:

DB2 trace

Statistics trace

Performance monitoring and tuning for data sharing environments (DB2 Data Sharing Planning and Administration)

DB2 in the z/OS environment (Introduction to DB2 for z/OS)

DB2 internal resource lock manager (Introduction to DB2 for z/OS)

Related information:

Statistics Report CPU times (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Investigating thread-level application performance

You can use DB2 accounting reports to analyze the class 2 CPU, class 3 suspension, and not accounted times, to investigate the performance of applications at the thread level.

Related tasks:

Programming applications for performance

Related information:

DB2 for z/OS System and Application Monitoring and Tuning

Narrowing your application performance investigation

When you encounter a general performance complaint or observe a response time problem, you can try to isolate the problem to a particular application or program, before you attempt system-wide tuning actions.

About this task

When you compare the response times for application programs, you can focus your comparison on Class 2 CPU time, Class 3 response time, and not-accounted times.

When your goal is general performance improvements, a good approach is to focus on the applications that use the most resources first. For example, you might focus your investigation on the top 10 plans or packages that meet the following criteria:

- Run most frequently.
- Use the most DB2 (class 2) CPU time.
- Have high elapsed times.

If you have performance history records, you can identify the transactions that show the largest increases.

Procedure

To narrow the scope of your investigation for general response time complaints:

1. Identify the plan or package that has the longest response times.
2. For plans that can potentially allocate many different static SQL packages, identify the package that has the longest response times.

   a. Use package-level accounting reports to determine which package has a long elapsed time.
   b. Use the class 7 CPU time for packages to determine which package has the largest CPU time or the greatest increase in CPU time.
   c. Enable class 10 to get detailed buffer manager, lock manager, and SQL statistics at the package level.
   d. Use class 8 for package elapsed time issues.
3. For dynamic SQL, use the following approaches to identify the SQL statement that use the most resources:
   - Use DB2 trace:
     – Check the global and local cache usage statistic. These statistics are externalized by IFCID 0002.
     – Use IFCID 0316 to find timestamp values, bind options, cumulative execution statistics and wait times.
     – Use IFCID 0317 to find the complete statement text and attribute string.
     – Activate IFCID 0318 to enable the capture IFCID 0316 and 0317 data.
   - Capture and analyze EXPLAIN information for the dynamic statement cache. You can issue EXPLAIN statements to capture access path information in the various EXPLAIN tables. You can issue the following statement to find access path information for each statement in the dynamic statement cache:

     ```
     EXPLAIN STMTCACHE ALL
     ```

     When you issue this statement, DB2 writes a single row for each statement in the dynamic statement cache to the DSN_STATEMENT_CACHE_TABLE only. No data is written to other EXPLAIN tables. You can use the STMT_ID and QUERYNO columns to correlate the DSN_STATEMENT_CACHE_TABLE rows with the other EXPLAIN tables.
4. Use the OMEGAMON SQL activity report to analyze specific SQL statements. You can also use OMEGAMON to analyze specific SQL statements, including the currently running SQL statement.
5. If you have a history of the performance of the affected application, compare current EXPLAIN output to previous access paths and costs.

Related concepts:

Response times

Accounting trace

Performance trace

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related tasks:

Collecting statement-level statistics for SQL statements

Monitoring the dynamic statement cache with READS calls

Capturing performance information for dynamic SQL statements

Related reference:

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Package identification in the accounting report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

SQL Activity Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Dynamic SQL Statement in the accounting report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Investigating class 2 CPU times

Class 2 CPU times indicate the amount of time consumed in DB2 on the central processor during the accounting interval. It does not include application time.

Procedure

To investigate high class 2 CPU times, complete the following investigations:

- Check whether unnecessary trace options are enabled. Excessive performance tracing can cause increased class 2 CPU times.
- Check the SQL statement count, the getpage count, and the buffer update count on the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report. If the profile of the SQL statements has changed significantly, review the application. If the getpage counts or the buffer update counts change significantly, check for changes to the access path and significant increases in the number of rows in the tables that are accessed.
- Check for the access paths problems with the SQL statements in your application.
- Use the statistics report to check buffer pool activity, including the buffer pool thresholds. If buffer pool activity has increased, ensure that your buffer pools are properly tuned.
- Check the counts in the locking section of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report. For a more detailed analysis, use the deadlock or timeout traces from statistics trace class 3 and the lock suspension report or trace. Latch contention is often a cause of high class 2 CPU times.

Related tasks:

Minimizing the volume of DB2 trace data

Tuning database buffer pools

Investigating access path problems

Maintaining DB2 database statistics

Maintaining data organization

Improving concurrency

Related reference:

SQL Activity Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Statistics Report and Trace Blocks (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Locking in the accounting report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Locking Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Investigating class 3 suspension time

When you encounter class 3 suspension time, you can use suspension values in the accounting reports to focus your investigation. *Class 3 suspension time* is the amount of wait time, which includes synchronous buffer pool I/O wait time, log I/O wait time, lock and latch wait time and other wait times.

About this task

Accounting class 3 data provides detailed information about the distribution of suspension times and related events.

Procedure

To investigate high class 3 suspension times, complete the following investigations:

- Check the individual types of suspensions in the “Class 3 Suspensions” section of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report.
- If lock/latch, drain lock, and claim release suspension times are high, focus your investigation on contention problems, and improving concurrency:

  IRLM lock/latch suspensions

  IRLM lock/latch suspension time is time spent waiting for locked resources, and latches that are used for internal serialization within IRLM. Examine the accounting records to determine whether the suspension time is caused by locks or latches. If the suspension is caused by locks, use performance trace classes 6 and 7. If the suspensions is caused by latches, check for the following conditions:

  - The IRLM trace is active.
  - The WLM dispatching priority of the IRLM address space is too low. It is best to use SYSSTC dispatching priority for the IRLM address space.
  - The IRLM is queried frequently by requests such as DISPLAY DATABASE LOCKS and MODIFY irlmproc,STATUS commands.
  - The DEADLOCK TIME value is to small and locking rates are high.
  - A large number of locks are held before an operation commits. If the MAX HELD LOCKS value in the accounting report is high, commit more frequently.

  DB2 latch suspension times

  DB2 *latch suspension time* indicates wait time for latches that are acquired internally within DB2 for short term serialization of resources such as storage and control block changes.
- For greater than expected wait times for synchronous I/O suspensions, complete the following investigations. *Synchronous I/O suspension time* is the total application wait time for synchronous I/Os. It is the total of database I/O and log write I/O. In the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report, check the values for `SYNCHRON. I/O`, `DATABASE I/O`, and `LOG WRITE I/O`. Database I/O and log I/O are not reported separately at the package level.
  - Check whether the I/O suspension time is high because of a large number of I/O suspensions, or because of long suspension times for each I/O. Long I/O suspension times probably indicate problems that require investigation outside of DB2, such as problems with the IOS component of z/OS, the channel, or the I/O subsystem.
  - Check whether the log I/O suspension times are high. If you see high values for log I/O suspension you can try to improve the log read performance and improve the log write performance.

- Check the getpage count to look for access path changes. If it has significantly increased, then an access path change might have occurred. However, if the getpage count remained about the same, but the number of I/Os increased significantly, the problem is not an access path change.

  If you have data from accounting trace class 8, the number of synchronous and asynchronous read I/Os is available for individual packages. Determine which package or packages have unacceptable counts for synchronous and asynchronous read I/Os. Activate performance trace classes 1, 2, and 3 so that Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS SQL activity reports can identify the SQL statement or cursor that is causing the problem.
- Check for a lower than expected buffer pool hit ratio.
  1. Look at the number of synchronous reads in the buffer pool that are associated with the plan.
  2. Look at the related buffer pool hit ratio. The buffer pool hit ration is meaningful only for objects that are accessed randomly.
  3. If the buffer pool size and the buffer pool hit ratio for random reads is small, consider the following actions:
     - Increase the buffer pool size. By increasing the buffer pool size, you might reduce the amount of synchronous database I/O and reduce the synchronous I/O suspension time.
     - You might also reduce the value of the sequential buffer pool threshold (VPSEQT). However this change might impact sequential processing

     By increasing the buffer pool size, you might reduce the amount of synchronous database I/O and reduce the synchronous I/O suspension time.
- Check for system-wide database buffer pool problems. You can also use buffer pool analyzer feature of Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Performance Expert or Performance Monitor to manage and optimize the buffer pools.
- Check the SQL ACTIVITY section of the accounting report, and compare that with previous data. Also, check the names of the packages being executed to determine if the pattern of programs being executed has changed.
- Use the DSNACCOX stored procedure to check for data organization problems. Disorganized data might prevent the use of sequential detection. You can run can invoke the REORG utility to resolve data organization problems.
- Check for RID pool failures. You can use the values of the `FAIL-NO STORAGE` (QXNSMIAP) and `FAIL-LIMIT EXCEEDED` (QXMRMIAP) fields under `RID LIST TOTAL` in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report.
- Check for system-wide problems in the EDM pool.

- If other read I/O time is high, check for problems with:
  - Prefetch I/O operations
  - Disk contention
  - Access path problems
  - Buffer pools that require tuning
- If other write I/O time is high, check for problems with:
  - The I/O path
  - Disk contention
  - Buffer pools that require tuning

- If the service task suspensions time is high, check open and close activity, and commit activity.

  Wait times for the following activities are the most common contributors to service task suspensions:

  - Phase 2 commit processing for updates, inserts, and deletes (UPDATE COMMIT - QWACAWTE). This value includes wait time for Phase 2 commit Log writes and database writes for LOB with LOG NO. For data sharing environments, it includes page P-locks unlocks for updated pages and GBP writes.
  - The OPEN/CLOSE service task. You can minimize this wait time by using two strategies. If the threshold set by the value of the DSMAX subsystem parameter is frequently reached, increase the value of the DSMAX subsystem parameter. If this threshold is reached, change CLOSE YES to CLOSE NO on data sets that are used by critical applications.
  - The SYSLGRNG recording service task.
  - The Data set extend/delete/define service task (EXT/DEL/DEF). You can minimize this wait time by defining larger primary and secondary disk space allocation for the table space.
  - Other service tasks (OTHER SERVICE TASK). Contributors to the other service tasks suspensions are likely to include time spent on the network for outgoing allied threads over TCP/IP connections, VSAM catalog updates, and parallel query cleanup. Other contributors are possible. The performance trace of the following IFCIDs provide useful information when the OTHER SERVICE TASK value is high:
    - 0170 and 0171
    - 0046, 0047, 0048, 0049, with 0050, when more detail is needed.
- Check whether suspension times are elongated because of reasons that are usually associated with not accounted time. For example, a long wait time might be encountered because of a short wait to obtain a lock followed by a much longer wait to be re-dispatched, because of the CPU load. In such cases, the entire wait might be recorded as class 3 time.

Related concepts:

Read operations and prefetch I/O

Suspensions and wait time

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Introduction to Buffer Pool Analyzer (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Buffer pool thresholds

Related tasks:

Analyzing concurrency

Improving concurrency

Investigating access path problems

Maintaining data organization

Managing RID pool size

Designing EDM storage space for performance

Related reference:

Times - Class 3 - Suspensions (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

DSNACCOX stored procedure (DB2 SQL)

SQL Activity Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Investigating DB2 not accounted time

When you encounter high DB2 class 2 not accounted times, you can focus your investigations on certain resources and activities.

About this task

DB2 *class 2 not accounted time* represents time that DB2 cannot account for. It is DB2 accounting class 2 elapsed time that is not recorded as class 2 CPU time or class 3 suspensions.

The following formula defines DB2 Class 2 Not Accounted Time when no parallelism is involved:

```
"DB2 Class 2 Not Accounted Time" = "DB2 Class 2 Elapsed time" -
("DB2 Class 2 CPU time" + "DB2 Class 3 suspension time")
```

Significant increases in class 2 not accounted time might be the result of any of the following conditions:

- Too much detailed online tracing by monitor programs that use OP*n* or OPX buffer destinations. This situation is usually the primary cause of high not-accounted-for time on systems that are not CPU-constrained.
- Running in a very high CPU utilization environment and waiting for CPU cycles, if DB2 work WLM service class goals are not set properly.
- Running in a high z/OS paging environment and waiting for storage allocation.
- If the specialty engines are highly utilized and the SYS1.PARMLIB(IEAOPT*xx*) member has IIPHONORPRIORITY=NO and IFAHONORPRIORITY=NO setting.
- Frequent gathering of dataset statistics (SMF 46 Type 2 records)
- DD consolidation (z/OS parm DDCONS=YES DETAIL) overhead - APAR II07124
- CF Lock Structure system managed DUPLEXing because DB2 is not informed about related suspension waits.
- Delays because of asynchronous processing of lock requests when the CF Lock structure is in a remote CF
- In very I/O intensive environments, the Media Manager might be running out of request blocks.
- Time spent waiting for parallel tasks to complete (when query parallelism is used for the query). This problem is often a result when the value of the PARAMDEG subsystem parameter is too high.
- HSM (Hierarchical Storage Manager) dataset recall.
- Waiting for requests to be returned from SNA DB2 Server.
- Data set open contention related to PCLOSET being too small.
- DB2 internal suspend and resume looping when several threads are waiting for the same resource.

Procedure

To investigate high DB2 class 2 not accounted times, complete the following investigations:

- Check for the following types of activities and wait times:
  - Paging activity, by using RMF reports.
  - Processor wait time, by using RMF reports.
  - Return wait time for requests to be returned from VTAM or TCP/IP, by using RMF reports.
  - The use of online performance monitors. You can sometimes reduce eliminate or significantly reduce high not account times by turning off or reducing the intensity of performance traces that are used by online monitors.
  - Wait time for completion of parallel tasks. A high not accounted time is acceptable if it is caused by wait time for completion of parallel tasks.
- Check the SER.TASK SWTCH field in the "Class 3 Suspensions" section of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports.

Related concepts:

Response times

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related reference:

z/OS RMF Report Analysis

Related information:

What is DB2 Accounting Class 2 Not Accounted Time?

Investigating access path problems

You can investigate whether a performance problem is caused by the access path for a particular SQL statement.

Before you begin

Narrow your investigation to a particular application program or SQL statement.

About this task

Many access path performance regressions can be resolved by making sure that appropriate, current, and accurate statistics are available for the database objects referenced by an SQL statement. Even for regressions that are triggered by a change in the DB2 maintenance level, the underlying cause is often related to inadequate statistics.

Procedure

To investigate access path problems, complete the following investigations:

1. Check the accuracy and completeness of statistics for the objects in the SQL statement. Inaccurate statistics often result in inaccurate access path cost estimates. Check the statistics that support your SQL statements before completing any other access path comparisons or investigations.

a. Check the accuracy of the basic statistics that are needed for all database objects. These statistics describe the size and organization of objects such as tables and indexes. Examples of these statistics include values from the following columns:
   - CARDF
   - NPAGESF
   - NLEAF
   - CLUSTERRATIOF
   - DATATREPEATFACTORF

   You can call the DSNACCOX stored procedure to discover whether to invoke the REORG or RUNSTATS utilities to maintain the health database objects. The statistics advisor feature of IBM Data Studio, IBM Data Server Manager, or InfoSphere Optim Query Workload Tuner also provides recommendations for these statements.

b. Check the status of the selectivity statistics for the particular SQL statement. These include correlation and distribution statistics. They are scenario-specific, support specific SQL statements, and are not routinely collected for all database objects. Examples of these statistics include:
   - COLCARDF, LOW2KEY, and HIGH2KEY column values
   - Single and multiple column frequency statistics
   - Single and multiple column histogram statistics
   - Multicolumn cardinalities (such as KEYCARD and COLGROUP column values)

   The statistics advisor feature that IBM Data Studio, IBM Data Server Manager, or InfoSphere Optim Query Workload Tuner provides is especially useful for getting recommendations for collecting these statistics.

2. Compare the filtering estimate that DB2 uses for access path selection to the actual filtering at run time. When the estimated and actual filtering differ, DB2 might choose a poorly performing access path because the cost estimates are inaccurate.

   a. Query the FILTER_FACTOR column of the DSN_PREDICAT_TABLE table to obtain the estimated filter factor for a predicate.

   b. To determine the actual filter factor, issue a query to determine the number of qualified rows. Divide the resulting value by the total number of rows in the table. For example, assume that a statement contains a `STAT_CD='A'` predicate. You might issue the following query to find the number of rows that qualify:

   ```
   SELECT COUNT(*)
   FROM T1
   WHERE STAT_CD='A'
   FOR FETCH ONLY WITH UR;
   ```

   The resulting count divided by the table cardinality is the actual filter factor.

   You can take the following actions to improve the estimated filter factor when it differs greatly from the actual filter factor at run time:
   - Gather frequency statistics, or histogram statistics, or both.
   - Rewrite predicates to take advantage of frequency and histogram statistics.
   - Use the REOPT(ALWAYS) or REOPT(ONCE) bind options to make the values of the parameter markers and host variables available at bind or prepare time.

3. If you have a history of the performance of the affected application, use the EXPLAIN output to compare the current and previous access paths and costs. You can also use the access plan graph feature of IBM Data Studio or IBM Data Server Manager to analyze and compare the SQL access paths.
4. Check whether indexes are used, how many matching columns are used, and whether the application used a different access path because an index was dropped.You can use the index advisor feature of InfoSphere Optim Query Workload Tuner to investigate index usage.
5. Examine the number and methods of join operations that are used to access the data.

Related concepts:

Query and application performance analysis (Introduction to DB2 for z/OS)

Predicate filter factors

Using EXPLAIN to understand the access path (Introduction to DB2 for z/OS)

Investigating SQL performance by using EXPLAIN

Related tasks:

Collecting data for access path performance problems (Collecting data)

Maintaining DB2 database statistics

Collecting statement-level statistics for SQL statements

Reoptimizing SQL statements at run time

Monitoring the dynamic statement cache with READS calls

Generating visual representations of access plans (IBM Data Studio)

Related reference:

DSNACCOX stored procedure (DB2 SQL)

Related information:

Tuning SQL with Optim Query Tuner, Part 1: Understanding access paths (IBM developerWorks)

Troubleshooting access-path-related performance problems

Collecting data for access path performance problems

For performance problems that are related to access path issues, you can collect a description of the problem symptoms, the output of the EXPLAIN statement for the query, related data definition statements, and catalog statistics.

Before you begin

- Ensure that the data is well organized and that complete accurate and current statistics are available for relevant database objects.
- Investigate the access path problem.
- Use an optimization tool such as IBM Data Studio, IBM Data Server Manager, or DB2 Query Workload Tuner for z/OS to try to resolve the query performance problem.
- To collect data to diagnose performance problems, you need the appropriate DB2 administrative authority.

About this task

Most access path performance issues and access path performance regressions can be resolved by ensuring that a complete, current, and accurate set of statistics from the RUNSTATS utility is available to DB2. Include the basic statistics that are needed for all database objects, and selectivity statistics that support the particular SQL statement.

You can also use query optimization tools, such as the IBM Data Studio or IBM Data Server Manager, to analyze and resolve many query performance problems. For example, you can use the statistics advisor function in IBM Data Studio or IBM Data Server Manager to determine which RUNSTATS utility jobs capture the necessary statistics. RUNSTATS utility jobs capture both basic object statistics and the selectivity statistics to support the particular SQL statement. The recommendation is to use one of these tools to try to resolve access path-related performance problems before you contact IBM Software Support.

If you contact IBM Software Support, you can provide information to help diagnose your access path problems.

Procedure

To collect access path diagnostic data to send to IBM Software Support:

1. Generate an EXPLAIN report of the query during a time period when the query performed slowly. For example, issue the following EXPLAIN statement. Replace *query-number* with the PLAN_TABLE rows for the query, and replace *problem-SQL-statement* with the SQL statement.

   ```
   EXPLAIN PLAN SET QUERYNO = query-number FOR

   problem-SQL-statement;
   ```

   You can issue the following SQL statement to create a report that describes the access path for the SQL statement:

   ```
   SELECT *
   FROM PLAN_TABLE
   WHERE QUERYNO = query-number
   ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
   ```

2. Locate an EXPLAIN report of the query during a time period when the query performed as expected.
3. Collect related data definition statements and catalog statistics that describe the environment for the SQL statement. You can collect and send this information to IBM Software Support by using IBM Data Studio, IBM Data Server Manager, or InfoSphere Optim Query Workload Tuner.

Related concepts:

Investigating SQL performance by using EXPLAIN

Related tasks:

Managing and preventing access path change

Gathering information about SQL statements for IBM Software Support

Gather information about the environments for SQL statements (DB2 Query Workload Tuner for z/OS)

Gathering information about the environments for SQL workloads (DB2 Query Workload Tuner for z/OS)

Related reference:

EXPLAIN (DB2 SQL)

RUNSTATS (DB2 Utilities)

Chapter 43. Response times

Response time is the amount of time that DB2 requires to process an SQL statement. To correctly monitor response time, you must understand how it is reported.

The following figure shows how some of the main measures relate to the flow of a transaction, including the relationship between user response time, DB2 accounting elapsed times and DB2 total transit time.

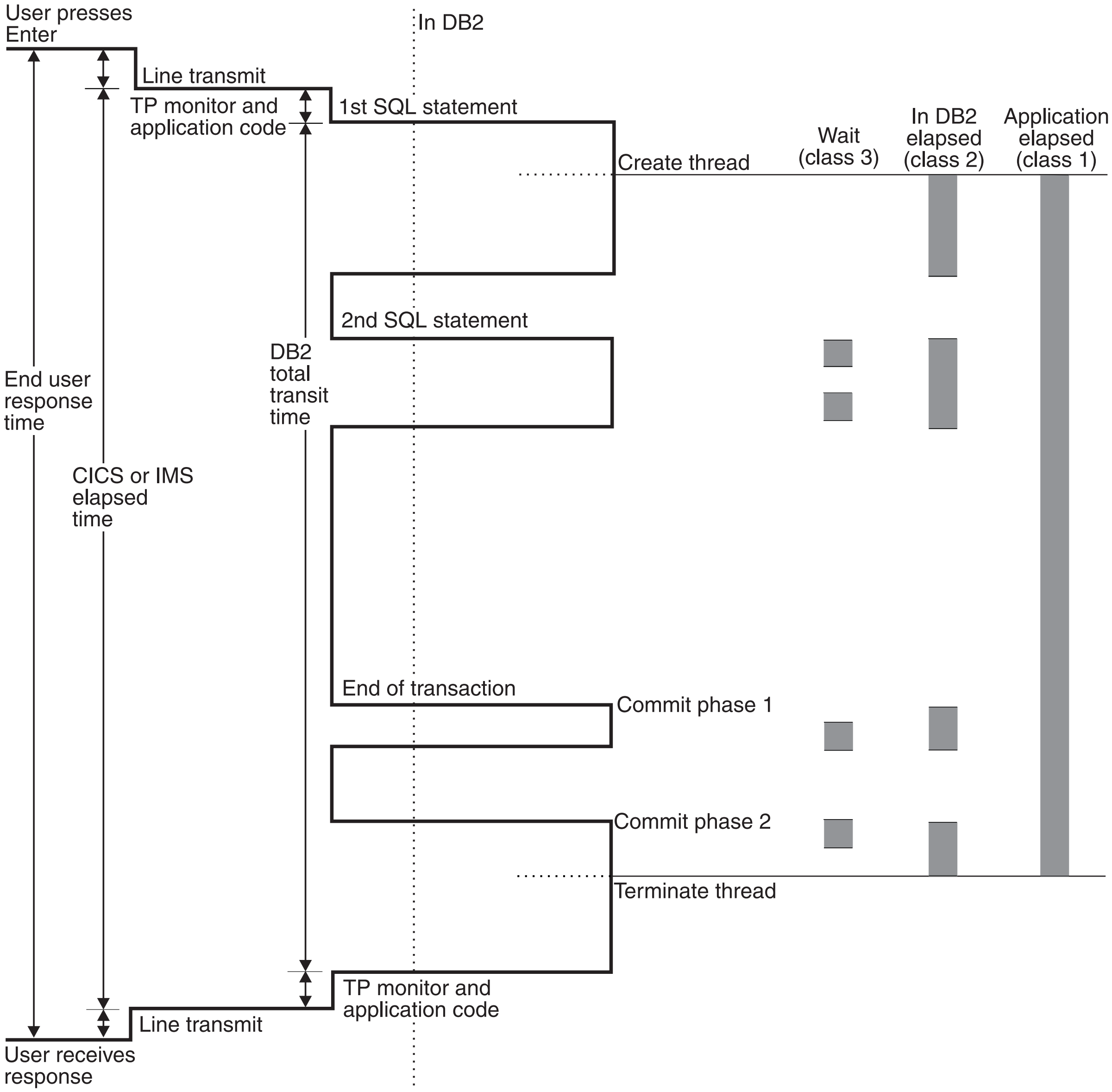


Figure 48. Transaction response times. Class 1 elapsed time includes application and DB2 elapsed times. Class 2 elapsed is the elapsed time in DB2. Class 3 is elapsed wait time in DB2. Standard accounting data is provided in IFCID 0003, which is turned on with accounting class 1. When accounting classes 2 and 3 are turned on as well, IFCID 0003 contains additional information about DB2 CPU times and suspension times.

End-user response time

End-user response time is the amount of time from the moment the end user presses the enter key until he or she receives the first response back at the terminal.

DB2 accounting elapsed times

DB2 *accounting elapsed times* are taken over the accounting interval. The start and end of the accounting interval differs based on factors such as the type of attachment facility, whether threads are reused, the value of the CMTSTAT subsystem parameter, and other factors. The elapsed times are collected in the records from the accounting trace and can be found in the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports.

Table 122. Accounting interval boundaries for different thread types

Attachment facility or thread type	Accounting interval details
CICS attachment facility	The accounting interval begins during thread creation processing. When CICS implements thread reuse, a change in the authorization ID or the transaction code initiates the sign-on process. This process terminates the accounting interval and creates the accounting record. TXIDSO=NO eliminates the sign-on process when only the transaction code changes. When a thread is reused without initiating sign-on, several transactions are accumulated into the same accounting record. The accumulated transactions can make it difficult to analyze a specific transaction occurrence and correlate DB2 accounting with CICS accounting. However, applications that use ACCOUNTREC(UOW) or ACCOUNTREC(TASK) in the DBENTRY RDO definition initiate a partial sign-on, which creates an accounting record for each transaction. You can use this data to tune your programs and to determineDB2 resource costs. **Related information:** Relating DB2 accounting records to CICS performance class records (CICS DB2 Guide)
IMS attachment facility	The accounting interval begins during thread creation processing. When an IMS thread is reused, the accounting interval ends and the accounting record is created at the next signon. When an IMS thread is not reused, the accounting interval ends at thread termination.

Table 122. Accounting interval boundaries for different thread types (continued)

Attachment facility or thread type	Accounting interval details
RRS attachment facility	The accounting interval begins during thread creation processing. The end of the accounting interval is controlled by the value specified for the *accounting-interval* area of the SIGNON function for RRSAF. **Related information:** SIGNON function for RRSAF (DB2 Application programming and SQL)
Distributed Data Facility threads	The accounting interval starts when the first SQL statement is received. The end of the accounting interval depends on the value of the CMTSTAT subsystem parameter, and whether the connection can go inactive at commit: **CMTSTAT=ACTIVE** The accounting interval ends at the end of processing for the thread. **CMTSTAT=INACTIVE** • If the thread can go inactive, the accounting interval ends at commit. • If the thread cannot go inactive for some reason, the accounting interval ends at the end of processing for the thread. • If the thread cannot go inactive specifically because the KEEPDYNAMIC(YES) bind option is in effect for one of the packages, the accounting interval ends at commit. **Related information:** DDF THREADS field (CMTSTAT subsystem parameter) (DB2 Installation and Migration) Methods for keeping prepared statements after commit points KEEPDYNAMIC bind option (DB2 Commands)
Other allied attached threads	The accounting interval begins during thread creation processing. The accounting interval ends at thread termination and includes a portion of the time spent terminating the thread.

The following types of elapsed times are provided:

Class 1 elapsed time
Class 1 elapsed time is the duration of the accounting interval. It includes time spent in DB2 as well as time spent in the front end. It is sometimes referred to as "application time."

Class 2 elapsed time
Class 2 elapsed time is the time spent in the DB2 thread during the accounting interval. It represents the sum of the times from any entry into DB2 until the corresponding exit from DB2. It is also sometimes referred to as the time spent "in DB2." It begins when the thread is created or, for reused threads, when another authorization ID signs on. It ends when the thread is terminated, or for reused threads, when another authorization ID signs on. This value is produced only when accounting class 2 trace is active. If class 2 trace is not active for the duration of the thread, the class 2 elapsed time does not reflect the entire DB2 time for the thread, but only the time when the class 2 trace was active.

Class 3 suspension time
Class 3 suspension time is the amount of wait time, which includes synchronous buffer pool I/O wait time, log I/O wait time, lock and latch wait time and other wait times.

Class 5 elapsed time
Class 5 elapsed time indicates the amount of elapsed time spent in DB2 processing instrumentation facility interface (IFI) requests. This time is included as part of the value for class 2 elapsed time.

Depending on your environment, additional considerations might be necessary. For example:

- Parallelism requires special considerations for accounting.
- Elapsed times for stored procedures, user-defined functions, or triggers are reported separately and are also included in the class 1 and class 2 total elapsed times.

DB2 class 2 not accounted time

DB2 *class 2 not accounted time* represents time that DB2 cannot account for. It is DB2 accounting class 2 elapsed time that is not recorded as class 2 CPU time or class 3 suspensions.

The following formula defines DB2 Class 2 Not Accounted Time when no parallelism is involved:

```
"DB2 Class 2 Not Accounted Time" = "DB2 Class 2 Elapsed time" -
("DB2 Class 2 CPU time" + "DB2 Class 3 suspension time")
```

Not-in-DB2 time

"Not-in-DB2" time is the calculated difference between the class 1 and the class 2 elapsed time. This value is the amount of time spent outside of DB2, but within the DB2 accounting interval. A lengthy time can be caused by thread reuse, which can increase class 1 elapsed time, or a problem in the application program, CICS, IMS, or the overall system.

For distributed applications, not-in-DB2 time is calculated with the following formula:

```
"Not-in-DB2" time = A - (B + C + (D - E))
```

Where the variables have the following values:

- *A* is the class 1 elapsed time.
- *B* is the class 2 non-nested elapsed time
- *C* is the class 1 non-nested elapsed time of any stored procedures, user-defined functions, or triggers
- *D* is class 1 non-nested CPU time
- *E* is class 2 non-nested CPU time

Related concepts:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related tasks:

Monitoring parallel operations

Related reference:

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

Analyzing DB2 response time (Tivoli Decision Support for z/OS)

Suspensions and wait time

Class 3 suspension time is the amount of wait time, which includes synchronous buffer pool I/O wait time, log I/O wait time, lock and latch wait time and other wait times.

IRLM lock/latch

IRLM lock/latch suspension time is time spent waiting for locked resources, and latches that are used for internal serialization within IRLM. Examine the accounting records to determine whether the suspension time is caused by locks or latches. If the suspension is caused by locks, use performance trace classes 6 and 7. If the suspensions is caused by latches, check for the following conditions:

- The IRLM trace is active.
- The WLM dispatching priority of the IRLM address space is too low. It is best to use SYSSTC dispatching priority for the IRLM address space.
- The IRLM is queried frequently by requests such as DISPLAY DATABASE LOCKS and MODIFY irlmproc,STATUS commands.
- The DEADLOCK TIME value is to small and locking rates are high.
- A large number of locks are held before an operation commits. If the MAX HELD LOCKS value in the accounting report is high, commit more frequently.

DB2 latch

DB2 *latch suspension time* indicates wait time for latches that are acquired internally within DB2 for short term serialization of resources such as storage and control block changes.

Synchronous I/O suspension time

Synchronous I/O suspension time is the total application wait time for synchronous I/Os. It is the total of database I/O and log write I/O. In the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report, check the values for `SYNCHRON. I/O`, `DATABASE I/O`, and `LOG WRITE I/O`. Database I/O and log I/O are not reported separately at the package level.

Service task suspensions

Service task suspension is the accumulated wait time from switching synchronous execution units, by which DB2 switches from one execution unit to another.

Wait times for the following activities are the most common contributors to service task suspensions:

- Phase 2 commit processing for updates, inserts, and deletes (UPDATE COMMIT - QWACAWTE). This value includes wait time for Phase 2 commit Log writes and database writes for LOB with LOG NO. For data sharing environments, it includes page P-locks unlocks for updated pages and GBP writes.
- The OPEN/CLOSE service task. You can minimize this wait time by using two strategies. If the threshold set by the value of the DSMAX subsystem parameter is frequently reached, increase the value of the DSMAX subsystem parameter. If this threshold is reached, change CLOSE YES to CLOSE NO on data sets that are used by critical applications.
- The SYSLGRNG recording service task.
- The Data set extend/delete/define service task (`EXT/DEL/DEF`). You can minimize this wait time by defining larger primary and secondary disk space allocation for the table space.
- Other service tasks (`OTHER SERVICE TASK`). Contributors to the other service tasks suspensions are likely to include time spent on the network for outgoing allied threads over TCP/IP connections, VSAM catalog updates, and parallel query cleanup. Other contributors are possible. The performance trace of the following IFCIDs provide useful information when the `OTHER SERVICE TASK` value is high:
  - 0170 and 0171
  - 0046, 0047, 0048, 0049, with 0050, when more detail is needed.

In the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report, the total of this information is reported in the `SER.TASK SWTCH` field. The field is the total of the five fields that follow it. If several types of suspensions overlap, the sum of their wait times can exceed the total clock time that DB2 spends waiting. Therefore, when service task suspensions overlap other types, the wait time for the other types of suspensions is not counted.

Other read suspensions

Other read suspensions result from by waiting to read pages that already have I/O in progress. The reported value is the accumulated wait time for read I/O for threads other than this one. It includes time for:

- Sequential prefetch
- List prefetch
- Dynamic prefetch
- Synchronous read I/O performed by a thread other than the one being reported

Other write suspensions

Other write suspensions result from waiting to update pages that already have I/O in progress. The reported value is the accumulated wait time for write I/O for threads other than this one. It includes time for asynchronous write I/O and synchronous write I/O performed by a thread other than the one being reported As a guideline, an asynchronous write I/O takes 0.1 to 2 milliseconds per page.

Page latch suspension

Page latch suspension indicates the accumulated wait time because of page latch contention.

Page latch contention can occur in highly concurrent insert environments. When a page is written to disk, multiple threads wait to update the same page, the first thread waits for other write I/O and other threads must wait for page latches. Longer page latch waits might occur if the disk writes are slower because of disk I/O performance issues. Page latch contention on data pages can occur during highly sequential updates to the same page from multiple threads. In a data sharing environment, high page latch contention might occur because of global locks for multithreaded applications that run on multiple members and use many insert, update, and delete operations.

- If the suspension is on the index leaf page, use one of the following strategies:
  - Make the inserts random
  - Drop the index
  - Perform the inserts from a single member
  - Use a smaller index page size
- If the page latch suspension is on a space map page, use the MEMBER CLUSTER option for the table space.
- Activate and analyze the performance trace for IFCIDs 0226 and 0227 to analyze the page latch details.

The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lock suspension report shows this suspension for page latch contention in the "other" category.

In the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report, this information is reported in the `PAGE LATCH` field.

Global lock suspension

Global transaction locks are used in data sharing environments. Their scope includes the entire data sharing group.

Related information:

Concurrency and locks in data sharing environments (DB2 Data Sharing Planning and Administration)

Improving concurrency in data sharing environments (DB2 Data Sharing Planning and Administration)

Options for reducing space map page contention (DB2 Data Sharing Planning and Administration)

Related concepts:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related tasks:

Investigating class 3 suspension time

Improving concurrency

Related reference:

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

You can obtain Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports of accounting data in long or short format and in various levels of detail.

The examples of Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports in this information are based on the default formats, which might have been modified for your installation. Furthermore, the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports have been reformatted or modified for this publication. You can also time results for nested activities such as triggers, stored procedures, and user-defined functions.

Use the long format accounting report for detailed analysis when you have isolated a problem to a particular application.

When you analyze the long format accounting report, you might focus your initial investigation on the following components of response time. Figure 49 on page 689 shows the arrangement of the following values in a sample report:

`APPL(CL.1) ELAPSED TIME`
Class 1 elapsed time is the duration of the accounting interval. It includes time spent in DB2 as well as time spent in the front end. It is sometimes referred to as "application time."

`DB2 (CL.2) ELAPSED TIME`
Class 2 elapsed time is the time spent in the DB2 thread during the accounting interval. It represents the sum of the times from any entry into DB2 until the corresponding exit from DB2. It is also sometimes referred to as the time spent "in DB2." It begins when the thread is created or, for reused threads, when another authorization ID signs on. It ends when the thread is terminated, or for reused threads, when another authorization ID signs on. This value is produced only when accounting class 2 trace is active. If class 2 trace is not active for the duration of the thread, the class 2 elapsed time does not reflect the entire DB2 time for the thread, but only the time when the class 2 trace was active.

`IFI (CL.5) ELAPSED TIME`
Class 5 elapsed time indicates the amount of elapsed time spent in DB2 processing instrumentation facility interface (IFI) requests. This time is included as part of the value for class 2 elapsed time.

`CLASS 3 SUSPENSIONS TOTAL CLASS 3`
Class 3 suspension time is the amount of wait time, which includes synchronous buffer pool I/O wait time, log I/O wait time, lock and latch wait time and other wait times.

`IRLM LOCK/LATCH`
IRLM lock/latch suspension time is time spent waiting for locked resources, and latches that are used for internal serialization within IRLM.

`DB2 LATCH`
DB2 *latch suspension time* indicates wait time for latches that are acquired internally within DB2 for short term serialization of resources such as storage and control block changes.

If the DB2 latch suspension time is high, check the statistics report data that reports the frequency of the DB2 latch contentions and identify the

DB2 latch classes (LC*nn*) (QVLSLC01 to QVLSLC32 and QVLSLC254) that have high rates. Look for tuning opportunities for any latches that exceed 10000 per second.

The following latch classes typically result in high contention:

LC06 Index leaf page split latch in data sharing.

LC14 Buffer pool lease recently used chain latch.

LC19 Log output buffer latch.

LC24 Prefetch latch or EDM least recently used chain latch.

`SYNCHRON. I/O`

Synchronous I/O suspension time is the total application wait time for synchronous I/Os. It is the total of database I/O and log write I/O. In the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report, check the values for `SYNCHRON. I/O`, `DATABASE I/O`, and `LOG WRITE I/O`. Database I/O and log I/O are not reported separately at the package level.

`OTHER READ I/O`

Other read suspensions result from by waiting to read pages that already have I/O in progress. The reported value is the accumulated wait time for read I/O for threads other than this one. It includes time for:

- Sequential prefetch
- List prefetch
- Dynamic prefetch
- Synchronous read I/O performed by a thread other than the one being reported

`OTHER WRITE I/O`

Other write suspensions result from waiting to update pages that already have I/O in progress. The reported value is the accumulated wait time for write I/O for threads other than this one. It includes time for asynchronous write I/O and synchronous write I/O performed by a thread other than the one being reported

`SER.TASK SWTCH`

Service task suspension is the accumulated wait time from switching synchronous execution units, by which DB2 switches from one execution unit to another.

Wait times for the following activities are the most common contributors to service task suspensions:

- Phase 2 commit processing for updates, inserts, and deletes (UPDATE COMMIT - QWACAWTE). This value includes wait time for Phase 2 commit Log writes and database writes for LOB with LOG NO. For data sharing environments, it includes page P-locks unlocks for updated pages and GBP writes.
- The OPEN/CLOSE service task. You can minimize this wait time by using two strategies. If the threshold set by the value of the DSMAX subsystem parameter is frequently reached, increase the value of the DSMAX subsystem parameter. If this threshold is reached, change CLOSE YES to CLOSE NO on data sets that are used by critical applications.
- The SYSLGRNG recording service task.

- The Data set extend/delete/define service task (`EXT/DEL/DEF`). You can minimize this wait time by defining larger primary and secondary disk space allocation for the table space.
- Other service tasks (`OTHER SERVICE TASK`). Contributors to the other service tasks suspensions are likely to include time spent on the network for outgoing allied threads over TCP/IP connections, VSAM catalog updates, and parallel query cleanup. Other contributors are possible. The performance trace of the following IFCIDs provide useful information when the `OTHER SERVICE TASK` value is high:
  - 0170 and 0171
  - 0046, 0047, 0048, 0049, with 0050, when more detail is needed.

`PAGE LATCH`

Page latch suspension indicates the accumulated wait time because of page latch contention.

Page latch contention can occur in highly concurrent insert environments. When a page is written to disk, multiple threads wait to update the same page, the first thread waits for other write I/O and other threads must wait for page latches. Longer page latch waits might occur if the disk writes are slower because of disk I/O performance issues. Page latch contention on data pages can occur during highly sequential updates to the same page from multiple threads. In a data sharing environment, high page latch contention might occur because of global locks for multithreaded applications that run on multiple members and use many insert, update, and delete operations.

- If the suspension is on the index leaf page, use one of the following strategies:
  - Make the inserts random
  - Drop the index
  - Perform the inserts from a single member
  - Use a smaller index page size
- If the page latch suspension is on a space map page, use the MEMBER CLUSTER option for the table space.
- Activate and analyze the performance trace for IFCIDs 0226 and 0227 to analyze the page latch details.

The Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lock suspension report shows this suspension for page latch contention in the "other" category.

`APPL(CL.1)CPU TIME`

Class 1 CPU times indicate the amount of time consumed in both DB2 and in applications on the central processor during the accounting interval.

`DB2(CL.2) CP CPU TIME`

Class 2 CPU times indicate the amount of time consumed in DB2 on the central processor during the accounting interval. It does not include application time.

`IFI (CL.5) CP CPU TIME`

Class 5 CPU time indicates the amount of time consumed on the central processor for processing instrumentation facility interface (IFI) requests during the accounting interval. This time is a subset of and included in the values for class 2 CPU time.

SE CPU *Specialty engine CPU time* indicates CPU time consumed on the specialty engines (zIIP and zAAP). This time is not included in the class 1 or class 2 CPU times.

NOT ACCOUNT

DB2 *class 2 not accounted time* represents time that DB2 cannot account for. It is DB2 accounting class 2 elapsed time that is not recorded as class 2 CPU time or class 3 suspensions.

Not-in-DB2-time

"Not-in-DB2" time is the calculated difference between the class 1 and the class 2 elapsed time. This value is the amount of time spent outside of DB2, but within the DB2 accounting interval. A lengthy time can be caused by thread reuse, which can increase class 1 elapsed time, or a problem in the application program, CICS, IMS, or the overall system.

For distributed applications, not-in-DB2 time is calculated with the following formula:

```
"Not-in-DB2" time = A - (B + C + (D - E))
```

Where the variables have the following values:

- *A* is the class 1 elapsed time.
- *B* is the class 2 non-nested elapsed time
- *C* is the class 1 non-nested elapsed time of any stored procedures, user-defined functions, or triggers
- *D* is class 1 non-nested CPU time
- *E* is class 2 non-nested CPU time

The following figure shows part of an example long format accounting report, including the arrangement of the fields that are described here.

```
AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.136869    0.022632    0.000429    LOCK/LATCH(DB2+IRLM)      0.000192      0.14    #OCCURRENCES    :  1568491
 NONNESTED      0.124535    0.010763         N/A     IRLM LOCK+LATCH          0.000188      0.04    #ALLIEDS        :        0
 STORED PROC    0.012321    0.011859         N/A     DB2 LATCH                0.000004      0.11    #ALLIEDS DISTRIB:        0
 UDF            0.000002    0.000000         N/A    SYNCHRON. I/O             0.010347      6.38    #DBATS          :  1474051
 TRIGGER        0.000010    0.000010         N/A     DATABASE I/O             0.009948      6.18    #DBATS DISTRIB. :    94440
                                                     LOG WRITE I/O            0.000399      0.19    #NO PROGRAM DATA:        0
CP CPU TIME     0.004499    0.004212    0.000333    OTHER READ I/O            0.003111      3.52    #NORMAL TERMINAT:       29
 AGENT          0.004499    0.004212         N/A    OTHER WRTE I/O            0.000002      0.00    #DDFRRSAF ROLLUP:    82834
  NONNESTED     0.002150    0.002102    0.000333    SER.TASK SWTCH            0.000291      0.05    #ABNORMAL TERMIN:        0
  STORED PRC    0.002345    0.002107         N/A     UPDATE COMMIT            0.000014      0.01    #CP/X PARALLEL. :        0
  UDF           0.000001    0.000000         N/A     OPEN/CLOSE               0.000120      0.01    #UTIL PARALLEL. :        0
  TRIGGER       0.000003    0.000003         N/A     SYSLGRNG REC             0.000004      0.00    #IO PARALLELISM :     1402
 PAR.TASKS      0.000000    0.000000         N/A     EXT/DEL/DEF              0.000015      0.00    #PCA RUP COUNT  :      N/A
                                                     OTHER SERVICE            0.000139      0.03    #RUP AUTONOM. PR:      N/A
SE CPU TIME     0.003034    0.002970         N/A    ARC.LOG(QUIES)            0.000000      0.00    #AUTONOMOUS PR  :      N/A
 NONNESTED      0.003032    0.002968         N/A    LOG READ                  0.000000      0.00    #INCREMENT. BIND:       40
 STORED PROC    0.000002    0.000002         N/A    DRAIN LOCK                0.000001      0.00    #COMMITS        :  1650053
 UDF            0.000000    0.000000         N/A    CLAIM RELEASE             0.000000      0.00    #ROLLBACKS      :     1457
 TRIGGER        0.000000    0.000000         N/A    PAGE LATCH                0.000002      0.16    #SVPT REQUESTS  :        0
                                                    NOTIFY MSGS               0.000002      0.00    #SVPT RELEASE   :        0
 PAR.TASKS      0.000000    0.000000         N/A    GLOBAL CONTENTION         0.000191      0.11    #SVPT ROLLBACK  :        0
                                                    COMMIT PH1 WRITE I/O      0.000000      0.00    MAX SQL CASC LVL:        3
SUSPEND TIME    0.000117    0.014248         N/A    ASYNCH CF REQUESTS        0.000038      0.52    UPDATE/COMMIT   :     1.54
 AGENT               N/A    0.014248         N/A    TCP/IP LOB XML            0.000071      0.05    SYNCH I/O AVG.  : 0.001623
 PAR.TASKS           N/A    0.000000         N/A    ACCELERATOR               0.000000      0.00
 STORED PROC    0.000116         N/A         N/A    AUTONOMOUS PROCEDURE           N/A       N/A
 UDF            0.000002         N/A         N/A    PQ SYNCHRONIZATION             N/A       N/A
                                                    TOTAL CLASS 3             0.014248     10.94
NOT ACCOUNT.         N/A    0.001201         N/A
DB2 ENT/EXIT         N/A        8.33         N/A
EN/EX-STPROC         N/A       46.96         N/A
EN/EX-UDF            N/A        0.00         N/A
DCAPT.DESCR.         N/A         N/A    0.000000
LOG EXTRACT.         N/A         N/A    0.000000
```

Figure 49. Partial accounting report (long format)

Related concepts:

Response times

Accounting for nested activities

Accounting trace

Related reference:

The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Accounting Short Report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Accounting Long Report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

Tivoli OMEGAMON XE for Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS

Correlating and synchronizing accounting records

You can match DB2 accounting records with CICS or IMS accounting records.

About this task

If a performance problem is outside of DB2, you can check the appropriate reports from a CICS or IMS reporting tool.

When CICS or IMS reports identify a commit, the timestamp can help you locate the corresponding Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting trace report.

Procedure

To simplify the task of correlating CICS and DB2 accounting records, use any of the following approaches:

- For reused threads (protected and unprotected threads), specify ACCOUNTREC(UOW) or ACCOUNTREC(TASK) on the DB2ENTRY RDO definition to help match CICS and DB2 accounting records. The CICS LU 6.2 token is included in the DB2 trace records, in field QWHCTOKN of the correlation header.
- Produce Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports that summarize accounting records by CICS transaction ID. Use the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS function Correlation Translation to select the subfield that contains the CICS transaction ID for reporting.
- Set the value of the STATIME subsystem parameter to 1. This setting specifies a one-minute interval for statistics collection. The more granular statistics can simplify the task of identifying short term spikes issues. Some records are always written at one-minute intervals, at the beginning of each minute, regardless of the value that you specify.
- Set the value of the SYNCVAL subsystem parameter to 0. This setting synchronizes statistics reporting with the SMF reporting interval, at the beginning of each hour (minute :00).

Related tasks:

Relating DB2 accounting records to CICS performance class records (CICS Transaction Server for z/OS)

Related reference:

STATISTICS TIME field (STATIME subsystem parameter) (DB2 Installation and Migration)

STATISTICS SYNC field (SYNCVAL subsystem parameter) (DB2 Installation and Migration)

Related information:

Online monitoring and reporting (Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS)

Chapter 44. Investigating SQL performance by using EXPLAIN

You capture detailed information about the access paths that DB2 chooses to process a statement, the cost of processing statements, and which functions DB2 uses.

PSPI

The information in EXPLAIN tables can help you to:

- Design databases, indexes, and application programs
- Determine when to rebind an application
- Determine the access path that DB2 chooses for a query

EXPLAIN data contains information about the access path that DB2 uses to process SQL statements. The primary use of EXPLAIN data is to investigate the access paths for the SELECT parts of your statements. For example, the data in EXPLAIN tables describes:

- Whether an index access or table space scan is used for each access to a table.
- When index access is used, how many indexes and index columns are used
- Which types of I/O methods are used to read the data pages.
- The join methods and types that are used, and the order in which DB2 joins the tables.
- When and why DB2 sorts data rows.

For UPDATE and DELETE WHERE CURRENT OF, and for INSERT, somewhat less information is provided. EXPLAIN data does not describe all or every type of access. For example, the access to LOB values, which are stored separately from the base table, and access to parent or dependent tables needed to enforce referential constraints, are not shown in EXPLAIN table data.

The access paths shown for the example queries are intended only to illustrate those examples. If you execute the same queries on your system, DB2 might choose different access paths.

PSPI

Tip: Various query optimization and administration tools, such as IBM Data Studio or IBM Data Server Manager and DB2 Query Workload Tuner for z/OS, provide a feature called *Visual Explain* that enables you to create visual diagrams of the access paths for your SQL statements. To learn more about using this feature, see Generating visual representations of access plans (IBM Data Studio).

Related tasks:

Collecting statement-level statistics for SQL statements

Monitoring the dynamic statement cache with READS calls

Investigating access path problems

Related reference:

EXPLAIN tables

EXPLAIN (DB2 SQL)

Creating EXPLAIN tables

Before you or optimization tools can capture and analyze EXPLAIN information, you must create the appropriate EXPLAIN tables to hold the information.

About this task

DB2 uses EXPLAIN tables to store information about the access plan that is uses to process SQL queries, and the tables are populated when the EXPLAIN function is invoked by you, by bind options, or by certain optimization tools. You can create one or more EXPLAIN tables that are qualified by a user ID for your own use and EXPLAIN tables that are qualified by SYSIBM.

Procedure

To create EXPLAIN tables:

Modify and run the appropriate sample job, or issue the CREATE TABLE statements, depending on the purpose of the tables:

Option	Description
EXPLAIN tables for your own use	Create a PLAN_TABLE and any additional plan tables qualified by a user ID by modifying the sample CREATE TABLE statements in the DSNTESC member of the SDSNSAMP library. Only PLAN_TABLE is required to enable the basic EXPLAIN function, but you can create additional tables to analyze the types of information that they capture.
EXPLAIN tables for use by optimization tools	EXPLAIN tables, qualified by SYSIBM, are created for use by SQL optimization tools when you run job DSNTIJSG to install and configure DB2-supplied routines. You might also be able to create the required tables, or specify an existing set of EXPLAIN tables, from the optimization tool's client interface.

Related tasks:

Converting EXPLAIN tables for migration from DB2 Version 8 (DB2 Installation and Migration)

Converting EXPLAIN tables (before migration) (DB2 Installation and Migration)

Migration step 24: Convert EXPLAIN tables to the current format and encoding type (DB2 Installation and Migration)

Related reference:

EXPLAIN tables

EXPLAIN table changes in DB2 10 (DB2 for z/OS What's New?)

Capturing access path information in EXPLAIN tables

You can populate EXPLAIN tables with information about the access paths that DB2 uses to process your SQL statements.

Before you begin

The following prerequisites must be met:

- EXPLAIN tables, including PLAN_TABLE, exist under the appropriate schema.
- You have the required authorities and privileges. For detailed information about the authorities and privileges, see EXPLAIN (DB2 SQL).

Procedure

To capture EXPLAIN information for SQL statements, use any of the following approaches:

- Issue an EXPLAIN statement. If aliases are defined on explain tables that were created with a different authorization ID, and you have the appropriate SELECT and INSERT privileges, you can populate the EXPLAIN tables even if you do not own them.

  You can issue the EXPLAIN statement statically from an application program, or dynamically by using QMF, SPUFI, or the command line processor.

  The process that DB2 uses to capture the access path information depends on options that you specify in the EXPLAIN statement. For example, if you specify the `PLAN FOR` *`explainable-statement`* option, DB2 uses the access path selection process to generate the access path information.

  However, if you specify the PACKAGE option, DB2 extracts information for the existing access paths that were selected for the statements when the package was bound. Similarly, if you specify the STMTCACHE option, DB2 extracts the information for the existing access paths that were selected when the statements were prepared and entered the dynamic statement cache.

- Specify the EXPLAIN(YES) option when you bind the plan or package. With EXPLAIN(YES), only a small amount of additional processing is required to insert the results in a plan table. The same processing for access path selection is performed, regardless of whether you use EXPLAIN(YES) or EXPLAIN (NO).

  If a plan or package that was previously bound with EXPLAIN(YES) is automatically rebound, the value of the ABEXP subsystem parameter determines whether EXPLAIN information is gathered again during the automatic rebind. Again, inserting the results into a plan table requires only a small amount of overhead. When you specify the EXPLAIN(YES) bind option, the information appears in table *package_owner*.PLAN_TABLE or *plan_owner*.PLAN_TABLE. For dynamically prepared SQL, the qualifier of PLAN_TABLE is the current SQLID.

  If the plan owner or the package owner has an alias on a PLAN_TABLE that was created by another owner, *other_owner*.PLAN_TABLE is populated instead of *package_owner*.PLAN_TABLE or *plan_owner*.PLAN_TABLE.

- Issue a BIND or REBIND command and specify the EXPLAIN(ONLY) option. The EXPLAIN tables are populated as if EXPLAIN(YES) was specified. However, the bind or rebind operation does not complete for the package. If the specified package already exists it is not dropped or replaced.

- For remote binds, you can specify EXPLAIN(YES) when binding a package at the server. You can use one of the following approaches:

- Specify EXPLAIN(YES) from the remote requester when binding a package at the DB2 server. The information appears in a plan table at the server, not at the requester.
- If the requester does not support the propagation of the EXPLAIN(YES) option, rebind the package at the requester and specify EXPLAIN(YES).

- Specify the CURRENT EXPLAIN MODE special register in the application. You can use this method to gather EXPLAIN information for dynamic statements, for a specific user and application, without any changes to the application logic.

Option	Description
CURRENT EXPLAIN MODE = NO	No EXPLAIN information is captured when explainable dynamic statements run. NO is the default value.
CURRENT EXPLAIN MODE = YES	Explainable dynamic SQL statements run normally, and information is captured to EXPLAIN tables after each statement is prepared and executed.
CURRENT EXPLAIN MODE = EXPLAIN	Explainable dynamic SQL statements do not execute, but information is captured to EXPLAIN tables after each statement is prepared in the application. Applications whose logic depends on the actual successful execution of statements fail if run with CURRENT EXPLAIN MODE = EXPLAIN. Only applications with simple application logic should use this option.

When YES or EXPLAIN are specified for the CURRENT EXPLAIN MODE special register, EXPLAIN information is captured during the prepare phase for packages bound with the REOPT(NONE) option and when the statement is reoptimized at run time for packages that are bound with the REOPT(ONCE | ALWAYS | AUTO) bind options.

For example, the following application program contains a dynamic statement, and EXPLAIN information is generated and inserted into the EXPLAIN tables after the statement is prepared and executed.

```
EXEC SQL DECLARE C1 CURSOR FOR PREP_STMT;
SOURCE_STMT = 'SELECT X, Y, Z FROM SCHEMA.TABLE1 WHERE X < Y ';
EXEC SQL SET CURRENT EXPLAIN MODE = YES;
EXEC SQL PREPARE PREP_STMT FROM SOURCE_STMT;
EXEC SQL OPEN C1;
```

- Invoke the DSNAEXP stored procedure. The DSNAEXP stored procedure is deprecated. PSPI

Related concepts:

Interpreting data access by using EXPLAIN

Related tasks:

Granting authorities for monitoring and tuning SQL statements

Related reference:

EXPLAIN tables

EXPLAIN (DB2 SQL)

EXPLAIN bind option (DB2 Commands)

DSNAEXP stored procedure (DB2 SQL)

EXPLAIN PROCESSING field (ABEXP subsystem parameter) (DB2 Installation and Migration)

CURRENT EXPLAIN MODE (DB2 SQL)

SET CURRENT EXPLAIN MODE (DB2 SQL)

SQLADM (Managing Security)

Capturing EXPLAIN information with QMF

You can use QMF to display the results of EXPLAIN to the terminal.

About this task

PSPI

You can create your own form to display the output or use the default form for QMF.

PSPI

Parameter markers in place of host variables

If you have host variables in a predicate for an original query in a static application and if you are using QMF or SPUFI to execute EXPLAIN for the query, you should consider using parameter markers where you use host variables in the original query.

PSPI

If you use a constant value instead, you might see different access paths for your static and dynamic queries. For instance, compare the queries in the following table:

Table 123. Three example queries for the use of parameter markers

Original Static SQL	QMF Query Using Parameter Marker	QMF Query Using Literal

`DECLARE CUR1`
`CURSOR FOR`
`SELECT * FROM T1`
`WHERE C1 > :HV`
	`EXPLAIN PLAN SET`
	`QUERYNO=1 FOR`
	`SELECT * FROM T1`
	`WHERE C1 > ?`
		`EXPLAIN PLAN SET`
		`QUERYNO=1 FOR`
		`SELECT * FROM T1`
		`WHERE C1 > 10`

Using the constant '10' would likely produce a different filter factor and might produce a different access path from the original static SQL statement. (A filter factor is the proportion of rows that remain after a predicate has "filtered out" the rows that do not satisfy the predicate. The parameter marker behaves just like a host variable, in that the predicate is assigned a default filter factor.

PSPI

Related concepts:

Predicate filter factors

Related reference:

EXPLAIN (DB2 SQL)

When to use a constant

If you know that a static plan or package was bound with REOPT(ALWAYS) or REOPT(AUTO) and you have some idea of what is returned in the host variable, including the constant in the QMF EXPLAIN results can be more accurate.

PSPI

REOPT(ALWAYS) means that DB2 replaces the value of the host variable with the true value at run time and then determine the access path. REOPT(AUTO) means that DB2 might replace the value of the host variable, depending on how it changed since the last execution..

PSPI

Related tasks:

Reoptimizing SQL statements at run time

Access path differences for static and dynamic SQL statements

Even when parameter markers are used, the access paths for static and dynamic queries might differ.

PSPI

DB2 assumes that the value that replaces a parameter marker has the same length and precision as the column that it is compared to. That assumption determines whether the predicate is stage 1 indexable or stage 2, which is always non-indexable.

If the column definition and the host variable definition are both strings, the predicate becomes stage 1 but not indexable when any of the following conditions are true:

- The column definition is CHAR or VARCHAR, and the host variable definition is GRAPHIC or VARGRAPHIC.
- The column definition is GRAPHIC or VARGRAPHIC, the host variable definition is CHAR or VARCHAR, and the length of the column definition is less than the length of the host variable definition.
- Both the column definition and the host variable definition are CHAR or VARCHAR, the length of the column definition is less than the length of the host variable definition, and the comparison operator is any operator other than "=".
- Both the column definition and the host variable definition are GRAPHIC or VARGRAPHIC, the length of the column definition is less than the length of the host variable definition, and the comparison operator is any operator other than "=".

The predicate becomes stage 2 when any of the following conditions are true:

- The column definition is DECIMAL(p,s), where p>15, and the host variable definition is REAL or FLOAT.
- The column definition is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC, and the host variable definition is DATE, TIME, or TIMESTAMP.

PSPI

Related concepts:

Differences between static and dynamic SQL (DB2 Application programming and SQL)

Variables in dynamic SQL (DB2 SQL)

Working with and retrieving EXPLAIN table data

DB2 and certain optimization tools automatically populate EXPLAIN tables, such as PLAN_TABLE and DSN_STATEMNT_TABLE, but you might decide to modify or remove data from certain instances of these tables.

About this task

Although DB2 automatically populates EXPLAIN tables, you might modify the data in your EXPLAIN tables. When DB2 populates EXPLAIN tables, or you ,INSERT triggers on the table are not activated. However, if you use an INSERT statement to add rows manually, then any INSERT triggers are activated.

Recommendation: Because certain optimization tools depend on the data in EXPLAIN tables, be careful not to manually modify data in EXPLAIN table instances created for those tools.

You can enable column and row-level access controls for EXPLAIN tables. However, the access controls do not apply when DB2 inserts data into the EXPLAIN tables.

Related concepts:

Interpreting data access by using EXPLAIN

Related tasks:

Correlating information across EXPLAIN tables

Monitoring SQL performance

Related reference:

EXPLAIN tables

Facilities and tools for DB2 performance monitoring

Retrieving EXPLAIN table rows for a plan

You can find the EXPLAIN table rows for all explainable statements of a particular plan in their logical order.

About this task

PSPI

Several processes can insert rows into the same plan table. To understand access paths, you must retrieve the rows for a particular query in the appropriate order. The rows for a particular plan are identified by the value of the APPLNAME column.

Procedure

To retrieve all the rows for all the explainable statements in a plan, in their logical order:

Issue the following SQL statement:

```
SELECT * FROM user-ID.PLAN_TABLE
  WHERE APPLNAME = 'application-name'
  ORDER BY EXPLAIN_TIME, QUERYNO[1], QBLOCKNO, PLANNO, MIXOPSEQ;
```

1. If the a static SQL package was bound with the EXPLAIN(YES) option and contains more than one statement with the same value for QUERYNO, use the SECTNOI column in place of QUERYNO.

Results

The result of the ORDER BY clause shows whether any of the following conditions exist:

- Multiple QBLOCKNO values within a QUERYNO or SECTNOI value
- Multiple PLANNO values within a QBLOCKNO value
- Multiple MIXOPSEQ values within a PLANNO value

All rows with the same non-zero value for QBLOCKNO and the same value for QUERYNO or SECTNOI relate to a step within the query. QBLOCKNO values are not necessarily executed in the order shown in PLAN_TABLE. But within a QBLOCKNO, the PLANNO column gives the sub-steps in the order they execute.

For each sub-step, the TNAME column identifies the table accessed. Sorts can be shown as part of a table access or as a separate step.

For entries that contain QUERYNO=0, use the EXPLAIN_TIME or SECTNOI column to distinguish individual statements.

PSPI

Retrieving EXPLAIN table rows for a package

You can retrieve the PLAN_TABLE rows for every explainable statement in a certain package.

About this task

PSPI

Several processes can insert rows into the same plan table. To understand access paths, you must retrieve the rows for a particular query in the appropriate order. The rows for a particular package are identified by the values of PROGNAME, COLLID, and VERSION. Those columns correspond to the four-part naming convention for packages:

```
location.collection.packageID.version
```

COLLID gives the COLLECTION name, and PROGNAME gives the PACKAGE_ID.

Procedure

To find the rows for all the explainable statements in a package, in their logical order:

Issue the following SQL statement:

```
SELECT * FROM userID.PLAN_TABLE
  WHERE PROGNAME = 'program-name'
    AND COLLID = 'collection-ID'
    AND VERSION = 'version-name'
  ORDER BY QUERYNO[1], QBLOCKNO, PLANNO, MIXOPSEQ;
```

1. If the a static SQL package was bound with the EXPLAIN(YES) option and contains more than one statement with the same value for QUERYNO, use the SECTNOI column in place of QUERYNO.

PSPI

Correlating information across EXPLAIN tables

When information about an SQL statement is captured into EXPLAIN tables, you can find relevant statistics and access path information from across the set of EXPLAIN tables. To find this information for a specific statement, use certain join predicates.

Procedure

To correlate information across the EXPLAIN tables:

1. Examine the STMT_TXT column of DSN_STATEMENT_CACHE_TABLE and find the corresponding values in the STMTID and CACHED_TS columns.
2. Use join predicates to correlate the various EXPLAIN tables. For EXPLAIN information that is generated by the EXPLAIN STMTCACHE statements:
   - Use the following predicates to join the DSN_STATEMENT_CACHE_TABLE and the PLAN_TABLE:

     ```
     DSN_STATEMENT_CACHE_TABLE.STMTID = PLAN_TABLE.QUERYNO AND
     DSN_STATEMENT_CACHE_TABLE.CACHED_TS = PLAN_TABLE.BIND_TIME
     ```
   - Use the following predicates to join the DSN_STATEMENT_CACHE_TABLE and other EXPLAIN tables, such a DSN_FUNCTION_TABLE or DSN_STATEMNT_TABLE:

     ```
     DSN_STATEMENT_CACHE_TABLE.STMTID = explain-table-name.QUERYNO AND
     DSN_STATEMENT_CACHE_TABLE.CACHED_TS = explain-table-name.EXPLAIN_TIME
     ```

   For EXPLAIN information that is generated by the EXPLAIN MODE special register:
   - Use the following predicate, which uses the EXPLAIN_TS column of DSN_STATEMENT_CACHE_TABLE instead of the CACHED_TS column, to join the DSN_STATEMENT_CACHE_TABLE and the PLAN_TABLE:

     ```
     DSN_STATEMENT_CACHE_TABLE.STMTID = PLAN_TABLE.QUERYNO AND
     DSN_STATEMENT_CACHE_TABLE.EXPLAIN_TS = PLAN_TABLE.BIND_TIME
     ```
   - Use the following predicate to join the DSN_STATEMENT_CACHE_TABLE and other EXPLAIN tables, such as DSN_FUNCTION_TABLE or DSN_STATEMNT_TABLE.

     ```
     DSN_STATEMENT_CACHE_TABLE.STMTID = explain-table-name.QUERYNO AND
     DSN_STATEMENT_CACHE_TABLE.EXPLAIN_TS = explain-table-name.EXPLAIN_TIME
     ```

Related concepts:

Dynamic SQL applications (Introduction to DB2 for z/OS)

Related tasks:

Capturing performance information for dynamic SQL statements

Improving dynamic SQL performance by enabling the dynamic statement cache

Related reference:

EXPLAIN tables

PLAN_TABLE

EXPLAIN (DB2 SQL)

EXPLAIN bind option (DB2 Commands)

Columns for correlating EXPLAIN tables

You can use a set of standard columns to correlate records for a particular static SQL statement from different EXPLAIN tables.

You can use the columns to uniquely identify and correlate all EXPLAIN records for a particular static SQL statement.

Every EXPLAIN table, except for DSN_STATEMENT_CACHE_TABLE, contains the following columns:

Table 124. Descriptions of standard columns for all EXPLAIN tables.

Column name	Data Type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.

Table 124. Descriptions of standard columns for all EXPLAIN tables. (continued)

Column name	Data Type	Description
APPLNAME	VARCHAR(24) NOT NULL[1]	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.

Table 124. Descriptions of standard columns for all EXPLAIN tables. (continued)

Column name	Data Type	Description
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.

Notes:

1. The data type of this column is VARCHAR(128) NOT NULL in the following tables:
   - DSN_COLDIST_TABLE
   - DSN_KEYTGTDIST_TABLE

Related tasks:

Correlating information across EXPLAIN tables

Creating EXPLAIN tables

Related reference:

EXPLAIN tables

Deleting EXPLAIN table rows

Although DB2 adds rows to EXPLAIN tables automatically. However, it does not automatically delete any rows from the tables.

About this task

As with other user tables, you can issue DELETE or TRUNCATE statements to remove data from PLAN_TABLE and the various related EXPLAIN tables.

You can use the QUERYNO, GROUP_MEMBER, and EXPLAIN_TIME columns to identify the corresponding rows in the various EXPLAIN tables. Other columns, such as APPLNAME and PROGNAME can also be used for this purpose.

When you consider your strategy for retaining EXPLAIN table data, remember that you might not need to keep EXPLAIN records for static SQL statements because you can issue EXPLAIN PACKAGE statements to recapture the EXPLAIN data at any time.

Important: Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Procedure

PSPI

To remove obsolete EXPLAIN table rows, you might use any of the following approaches:

- Use DELETE statements to remove rows from EXPLAIN tables based on the age of the rows. This approach is most appropriate for use in development environments. For example, you might use the following statement to delete all rows that are more than one month old from a particular EXPLAIN table:

  ```
  DELETE FROM table-name WHERE EXPLAIN_TIME < CURRENT TIMESTAMP - 1 MONTH;
  ```

- Establish a practice for selectively deleting obsolete or unneeded PLAN_TABLE rows, and then use DELETE statements to selectively remove rows from the related EXPLAIN tables, based on the PLAN_TABLE rows that remain. For example, the following statement deletes rows from DSN_DETCOST_TABLE that do not have corresponding rows in PLAN_TABLE:

  ```
  DELETE FROM DSN_DETCOST_TABLE DT
  WHERE NOT EXISTS
  (SELECT 1
   FROM PLAN_TABLE PT
   WHERE PT.QUERYNO = DT.QUERYNO
   AND PT.APPLNAME = DT.APPLNAME
   AND PT.PROGNAME = DT.PROGNAME
   AND PT.GROUP_MEMBER = DT.GROUP_MEMBER
   AND PT.EXPLAIN_TIME = DT.EXPLAN_TIME);
  ```

  In this example, the APPLNAME and PROGNAME columns are included to improve the performance of the subquery if many duplicate QUERYNO values exist. With this approach, take the following actions to avoid performance problems such as lock contention:

  - Ensure that an index exists on the following PLAN_TABLE columns: QUERYNO, APPLNAME, PROGNAME, EXPLAIN_TIME, GROUP_MEMBER. The PLAN_TABLE_HINT_IX index is suitable for this purpose. For more information about creating the PLAN_TABLE_HINT_IX index, see Preparing to influence access paths.
  - Lock each EXPLAIN table before issuing the DELETE statement for that table.
  - Issue a COMMIT statement after the DELETE statement for each table.
- Issue DROP statements for the EXPLAIN tables and create a new set of EXPLAIN tables. If you use this approach, no existing EXPLAIN table data is retained.

  PSPI

Related tasks:

Creating EXPLAIN tables

Related reference:

EXPLAIN tables

Columns for correlating EXPLAIN tables

DELETE (DB2 SQL)

TRUNCATE (DB2 SQL)

DROP (DB2 SQL)

Interpreting data access by using EXPLAIN

You can use the values captured in EXPLAIN tables, and instances of PLAN_TABLE in particular, to analyze how DB2 accesses data when it processes a particular SQL statement.

Tip: Various query optimization and administration tools, such as IBM Data Studio or IBM Data Server Manager and DB2 Query Workload Tuner for z/OS, provide a feature called *Visual Explain* that enables you to create visual diagrams of the access paths for your SQL statements. To learn more about using this feature, see Generating visual representations of access plans (IBM Data Studio).

Related concepts:

Investigating SQL performance by using EXPLAIN

Related tasks:

Investigating access path problems

Generating visual representations of access plans (IBM Data Studio)

Related reference:

EXPLAIN tables

EXPLAIN (DB2 SQL)

EXPLAIN bind option (DB2 Commands)

Questions for investigating data access

You can focus your investigation of access paths that DB2 uses to process SQL statements by using the PLAN_TABLE data to answer certain questions.

PSPI

Tip: You can use query optimization tools, such as IBM Data Studio or IBM Data Server Manager and DB2 Query Workload Tuner for z/OS, to capture EXPLAIN information and generate automated recommendations for how to improve the performance of your queries. For more information about using these tools, see Tuning single SQL statements (IBM Data Studio). For example, these products include a feature called *Visual Explain* that enables you to create visual diagrams of the access paths for your SQL statements. To learn more about using this feature, see Generating visual representations of access plans (IBM Data Studio).

However, if you do analyze PLAN_TABLE data directly, you can use the following questions to guide your initial analysis of the access paths:

How are indexes used to access the data?
The ACCESSTYPE and MATCHOLS values contain information about the use of indexes in an access path.

1. Is an index used? For information about interpreting index access, see “Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX')” on page 711.
2. How many indexes are used? For information about interpreting access through multiple indexes, see Multiple index access (ACCESSTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU').
3. How many index columns are used in matching? For more information about finding the number of matching index columns, see Matching index scan (MATCHCOLS>0) and “The number of index columns used for matching (MATCHCOLS=*n*)” on page 713.
4. Is the query satisfied by the index alone? For more information about analyzing index-only access, see Index-only access (INDEXONLY='Y').
5. How many index screening columns are used? For more information about analyzing index screening, see Index screening.

Is direct row access used?

Direct row access can only be used only when the table contains a column of the ROWID data type. For information about direct row access, see “Direct row access (PRIMARY_ACCESSTYPE='D')” on page 726 and ROWID data type (Introduction to DB2 for z/OS).

What possibly costly operations are used?

1. Is a view or nested table expression materialized? For more information about analyzing materialization, see View and nested table expression access.
2. Was a scan limited to certain partitions? For information about analyzing the use of page-range screening, see Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U').
3. What prefetch type is expected? For information about analyzing the use of prefetch, see “Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U')” on page 732.
4. Is data accessed or processed in parallel? For information about analyzing the use of parallelism in the access path, see Parallel processing access (PARALLELISM_MODE='I', 'C', or 'X').
5. Is data sorted? For information about analyzing the use of sort operations, see “Sort access” on page 735.
6. Is a subquery transformed to a join? For information about analyzing subquery access, see “Subquery access” on page 761.
7. When are aggregate functions evaluated? For information about analyzing when DB2 evaluates aggregate functions, see “Aggregate function access (COLUMN_FN_EVAL)” on page 709.
8. Is a complex trigger WHEN clause used? For information about identifying the use of triggers in a WHEN clause, see Complex trigger WHEN clause access (QBLOCKTYPE='TRIGGR').

What are the object dependencies for the access path?

Access paths have dependencies on the objects that are identified in the TNAME and ACCESSNAME columns of the PLAN_TABLE. However, access paths might also depend on objects that DB2 does not actually use when it processes the selected access paths. Such dependencies are not be shown in EXPLAIN output, but they are recorded in the SYSIBM.SYSPACKDEP table (DB2 SQL).

PSPI

Related concepts:
Interpreting data access by using EXPLAIN

Related tasks:

Generating visual representations of access plans (IBM Data Studio)

Related reference:
PLAN_TABLE

EXPLAIN (DB2 SQL)

Table space scan access (ACCESSTYPE='R' and PREFETCH='S')

DB2 sometimes needs to use a *table space scan* to access the data, usually because index access is not available for some reason.

PSPI

Table space scan is most often used for one of the following reasons:

- Access is through a created temporary table. (Index access is not possible for created temporary tables.)
- A matching index scan is not possible because an index is not available, or no predicates match the index columns.
- A high percentage of the rows in the table is returned. In this case, an index is not really useful because most rows need to be read anyway.
- The indexes that have matching predicates have low cluster ratios and are therefore efficient only for small amounts of data.

In some cases, a table space scan is used in combination with a sparse index. A sparse index is created from the initial table space scan, and subsequent access to the table from the same statement uses the sparse index instead of repeatedly scanning the table.

Example

Assume that table T has no index on C1. The following is an example that uses a table space scan:

```
SELECT * FROM T WHERE C1 = VALUE;
```

In this case, at least every row in table T must be examined to determine whether the value of C1 matches the given value.

Table space scans of nonsegmented table spaces

DB2 reads and examines every page in the table space, regardless of which table the page belongs to. It might also read pages that have been left as free space and space not yet reclaimed after deleting data.

Table space scans of segmented table spaces

If the table space is segmented, DB2 first determines which segments need to be read. It then reads only the segments in the table space that contains rows of table

T. If the prefetch quantity, which is determined by the size of your buffer pool, is greater than the SEGSIZE and if the segments for table T are not contiguous, DB2 might read unnecessary pages. Use a SEGSIZE value that is as large as possible, consistent with the size of the data. A large SEGSIZE value is best to maintain clustering of data rows. For very small tables, specify a SEGSIZE value that is equal to the number of pages required for the table.

The following table summarizes the recommended values for SEGSIZE, depending on how large the table is.

Table 125. Recommendations for SEGSIZE

Number of pages	SEGSIZE recommendation
≤ 28	4 to 28
> 28 < 128 pages	32
≥ 128 pages	64

Table space scans of partitioned table spaces

A table space scan on a partitioned table space can be more efficient than on a nonpartitioned table space because DB2 can use page range screening to limit access to only the required partitions. This type of access, which is sometimes called limited partition scan, requires predicates on the partitioning columns.

Table space scans and sequential prefetch

Regardless of the type of table space, DB2 plans to use sequential prefetch for a table space scan. For a segmented table space, DB2 might not actually use sequential prefetch at execution time if it can determine that fewer than four data pages need to be accessed.

If you do not want to use sequential prefetch for a particular query, consider adding the OPTIMIZE FOR 1 ROW clause to the query.

PSPI

Related concepts:

Page range screening (PAGE_RANGE='Y')

Sequential prefetch (PREFETCH='S')

Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U')

Related tasks:

Optimizing retrieval for a small set of rows (DB2 Application programming and SQL)

Related reference:

optimize-clause (DB2 SQL)

Aggregate function access (COLUMN_FN_EVAL)

The access path that is chosen for the SQL statement determines when DB2 evaluates aggregate functions.

PSPI

- If the ACCESSTYPE column value is I1, then a MAX or MIN function can be evaluated by one access of the index that is named in ACCESSNAME.
- For other values of ACCESSTYPE, the COLUMN_FN_EVAL column tells when DB2 is evaluating the aggregate functions.

Value	**Functions are evaluated ...**
S	During a sort to satisfy a GROUP BY clause
R	When data is being read from the table or index
Blank	After data retrieval and after any sorts

Generally, values of R and S are considered better for performance than a blank.

Care is required with use of the VARIANCE and STDDEV functions because they are always evaluated late (that is, COLUMN_FN_EVAL is blank). As a result, other functions in the same query block must be evaluated late too. For example, in the following query, the SUM function is evaluated later than it would be if the variance function was not present:

```
SELECT SUM(C1), VARIANCE(C1) FROM T1;
```

PSPI

Related concepts:

Aggregate functions (DB2 SQL)

Related reference:

group-by-clause (DB2 SQL)

MAX (DB2 SQL)

MIN (DB2 SQL)

STDDEV or STDDEV_SAMP (DB2 SQL)

VARIANCE or VARIANCE_SAMP (DB2 SQL)

Hash access (ACCESSTYPE='H', 'HN', or 'MH')

The ACCESSTYPE column in the plan table has a value of 'H', 'HN', or 'MH', when the table being accessed is organized by hash, and hash access is used to access the data.

PSPI

Hash access is efficient for queries that use equal predicates to access a single row on a table. Hash access also reduces CPU load. However the use of hash access requires additional storage space for maintenance of the hash space. If a table is organized by hash, index clustering is unavailable on that table.

When a hash access path is selected for a parallel group, parallelism is not selected for that parallel group, however DB2 might select parallelism for other parallel groups in the query.

When star join is enabled, DB2 does not use hash access for either the fact or dimension tables when a query contains a qualified star join.

PSPI

Related concepts:

Star join access (JOIN_TYPE='S')

Related tasks:

Organizing tables by hash for fast access to individual rows

Managing space and page size for hash-organized tables

Index access (ACCESSTYPE is 'I', 'IN', 'I1', 'N', 'MX', or 'DX')

If the ACCESSTYPE column in the plan table has a value of 'I', 'I1', 'IN', 'N', 'MX', or 'DX', DB2 uses an index to access the table that is named in column TNAME.

Introductory concepts

Creation of indexes (Introduction to DB2 for z/OS)

PSPI

The columns ACCESSCREATOR and ACCESSNAME identify the index.

If a nested loop join is used in processing the query, you might see ACCESSTYPE='R', but the value of the PRIMARY_ACCESSTYPE column is T. This indicates that sparse index access is used.

Indexes can provide efficient access to data. In fact, that is the only purpose of non-unique indexes. Unique indexes have the additional purpose of ensuring that key values are unique.

Special cases

- For dynamic SQL queries, DB2 avoids choosing indexes in which all of the partitions of the index are in a restricted state. If only some partitions are in a restricted state, an index might be chosen, because subsequent access might require only unrestricted partitions to be touched. This behavior allows an efficient index to be available as long as there is a possibility that it could be used successfully. For static queries, DB2 does not consider the state of the index partitions when choosing an index.
- DB2 might also use sparse index access (ACCESSTYPE='R' and PRIMARY_ACCESSTYPE='T') when processing a nested-loop join.

PSPI

Related tasks:

Designing indexes for performance

Related information:

Implementing DB2 indexes (DB2 Administration Guide)

Matching index scan (MATCHCOLS>0)

In a *matching index scan*, predicates are specified on either the leading or all of the index key columns. These predicates provide *filtering*; only specific index pages and data pages need to be accessed. If the degree of filtering is high, the matching index scan is efficient.

About this task

PSPI

In the general case, the rules for determining the number of matching columns are simple, but with a few exceptions.

- Look at the index columns from leading to trailing. For each index column, search for an indexable boolean term predicate on that column. (See Predicates and access path selection for a definition of boolean term.) If such a predicate is found, then it can be used as a matching predicate.

  Column MATCHCOLS in a plan table shows how many of the index columns are matched by predicates.
- If no matching predicate is found for a column, the search for matching predicates stops.
- If a matching predicate is a range predicate, then there can be no more matching columns. For example, in the matching index scan example that follows, the range predicate `C2>1` prevents the search for additional matching columns.
- For star joins, a missing key predicate does not cause termination of matching columns that are to be used on the fact table index.

The exceptional cases are:

- For MX, or DX accesses and index access with list prefetch, IN-list predicates cannot be used as matching predicates.
- Join predicates cannot qualify as matching predicates when doing a merge join (METHOD=2). For example, `T1.C1=T2.C1` cannot be a matching predicate when doing a merge join, although any local predicates, such as `C1='5'` can be used.

  Join predicates can be used as matching predicates on the inner table of a nested loop join or hybrid join.
- The XML index, containing composite key values, maps XML values to DOCID and NODEID pairs. The XML values (the first key value) in the composite keys can be specified in XPath expressions. By matching the XPath expression in an XMLEXISTS predicate to the XPath expression in a particular XML index, the index key entries which contain the matched key values can be identified. The DOCID and NODEID pairs of those identified index key entries can be used to locate the corresponding base table rows efficiently.

Matching index scan example

Assume an index was created on T(C1,C2,C3,C4):

```
SELECT * FROM T
  WHERE C1=1 AND C2>1
    AND C3=1;
```

Two matching columns occur in this example. The first one comes from the predicate `C1=1`, and the second one comes from `C2>1`. The range predicate on C2 prevents C3 from becoming a matching column.

PSPI

Related concepts:

Access methods with XML indexes (DB2 Programming for XML)

The number of index columns used for matching (MATCHCOLS=*n*):

If MATCHCOLS is 0, the access method is called a *nonmatching index scan* and all the index keys and their RIDs are read. If MATCHCOLS is greater than 0, the access method is called a *matching index scan* and the query uses predicates that match the index columns.

Introductory concepts

Creation of indexes (Introduction to DB2 for z/OS)

PSPI

In general, the matching predicates on the leading index columns are equal or IN predicates. The predicate that matches the final index column can be an equal, IN, NOT NULL, or range predicate (<, <=, >, >=, LIKE, or BETWEEN).

The following example illustrates matching predicates:

```
SELECT * FROM EMP
  WHERE JOBCODE = '5' AND SALARY > 60000 AND LOCATION = 'CA';

INDEX XEMP5 on (JOBCODE, LOCATION, SALARY, AGE);
```

The index XEMP5 is the chosen access path for this query, with MATCHCOLS = 3. Two equal predicates are on the first two columns and a range predicate is on the third column. Though the index has four columns in the index, only three of them can be considered matching columns.

PSPI

Related concepts:

Indexable and non-indexable predicates

Indexes on table columns (DB2 Administration Guide)

Related reference:

PLAN_TABLE

Summary of predicate processing

Index screening

In *index screening*, predicates are specified on index key columns but are not part of the matching columns.

Introductory concepts

Creation of indexes (Introduction to DB2 for z/OS)

PSPI

Those predicates improve the index access by reducing the number of rows that qualify while searching the index. For example, with an index on `T(C1,C2,C3,C4)` in the following SQL statement, `C3>0` and `C4=2` are index screening predicates.

```
SELECT * FROM T
  WHERE C1 = 1
    AND C3 > 0 AND C4 = 2
    AND C5 = 8;
```

The predicates can be applied on the index, but they are not matching predicates. However, `C5=8` is not an index screening predicate, and DB2 must evaluated that predicate when data is retrieved. The value of the MATCHCOLS column of PLAN_TABLE is 1.

The STAGE column of DSN_FILTER_TABLE identifies predicates that DB2 uses for index screening.

PSPI

Related concepts:

Indexable and non-indexable predicates

Indexes on table columns (DB2 Administration Guide)

Related reference:

DSN_FILTER_TABLE

PLAN_TABLE

Summary of predicate processing

Nonmatching index scan (ACCESSTYPE='I' and MATCHCOLS=0)

In a *nonmatching index scan* no matching columns are in the index. Consequently, all of the index keys must be examined.

PSPI

Because a nonmatching index usually provides no filtering, only a few cases provide an efficient access path. The following situations are examples:

When index screening predicates exist
In that case, not all of the data pages are accessed.

When the clause OPTIMIZE FOR *n* ROWS is used
That clause can sometimes favor a nonmatching index, especially if the index gives the ordering of the ORDER BY clause or GROUP BY clause.

When more than one table exists in a nonsegmented table space
In that case, a table space scan reads irrelevant rows. By accessing the rows through the nonmatching index, fewer rows are read.

PSPI

Related concepts:

Predicates and access path selection

Related tasks:

Minimizing the cost of retrieving few rows

Related reference:

Summary of predicate processing

PLAN_TABLE

optimize-clause (DB2 SQL)

Multiple index access (ACCESSTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU')

Multiple index access uses more than one index to access a table.

PSPI

Multiple index access is a good access path when:
- No single index provides efficient access.
- A combination of index accesses provides efficient access.

RID lists are constructed for each of the indexes involved. The unions or intersections of the RID lists produce a final list of qualified RIDs that is used to retrieve the result rows, using list prefetch. You can consider multiple index access as an extension to list prefetch with more complex RID retrieval operations in its first phase. The complex operators are union and intersection.

DB2 does not use multiple-index access for IN-list predicates.

DB2 chooses multiple index access for the following query:

```
SELECT * FROM EMP
   WHERE (AGE = 34) OR
         (AGE = 40 AND JOB = 'MANAGER');
```

For this query:
- EMP is a table with columns EMPNO, EMPNAME, DEPT, JOB, AGE, and SAL.
- EMPX1 is an index on EMP with key column AGE.
- EMPX2 is an index on EMP with key column JOB.

The plan table contains a sequence of rows describing the access. For this query, ACCESSTYPE uses the following values:

Value	Meaning
M	Start of multiple index access processing
MX	Indexes are to be scanned for later union or intersection
MI	An intersection (AND) is performed
MU	A union (OR) is performed

The following steps relate to the previous query and the values shown for the plan table in the following table:

1. Index EMPX1, with matching predicate AGE = 40, provides a set of candidates for the result of the query. The value of MIXOPSEQ is 1.
2. Index EMPX2, with matching predicate JOB = 'MANAGER', also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 2.
3. The first intersection (AND) is done, and the value of MIXOPSEQ is 3. This MI removes the two previous candidate lists (produced by MIXOPSEQs 2 and 3) by intersecting them to form an intermediate candidate list, IR1, which is not shown in PLAN_TABLE.
4. Index EMPX1, with matching predicate AGE = 34, also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 4.
5. The last step, where the value MIXOPSEQ is 5, is a union (OR) of the two remaining candidate lists, which are IR1 and the candidate list produced by MIXOPSEQ 1. This final union gives the result for the query.

Table 126. Plan table output for a query that uses multiple indexes. Depending on the filter factors of the predicates, the access steps can appear in a different order.

PLAN-NO	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	PREFETCH	MIXOP-SEQ
1	EMP	M	0		L	0
1	EMP	MX	1	EMPX1		1
1	EMP	MX	1	EMPX2		2
1	EMP	MI	0			3
1	EMP	MX	1	EMPX1		4
1	EMP	MU	0			5

The multiple index steps are arranged in an order that uses RID pool storage most efficiently and for the least amount of time.

Execution order for multiple index access

A set of rows in the plan table contain information about the multiple index access. The rows are numbered in column MIXOPSEQ in the order of execution of steps in the multiple index access. (If you retrieve the rows in order by MIXOPSEQ, the result is similar to postfix arithmetic notation.)

Both of the following examples have these indexes: IX1 on T(C1) and IX2 on T(C2).

Example: index access order

Suppose that you issue the following SELECT statement:

```
SELECT * FROM T
  WHERE C1 = 1 AND C2 = 1;
```

DB2 processes the query by performing the following steps:

1. DB2 retrieves all the qualifying record identifiers (RIDs) where C1=1, by using index IX1.
2. DB2 retrieves all the qualifying RIDs where C2=1, by using index IX2. The intersection of these lists is the final set of RIDs.
3. DB2 accesses the data pages that are needed to retrieve the qualified rows by using the final RID list.

The plan table for this example is shown in the following table.

Table 127. PLAN_TABLE output for example with intersection (AND) operator

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	M	0		N	L	0
T	MX	1	IX1	Y		1
T	MX	1	IX2	Y		2
T	MI	0		N		3

Example: multiple access to the same index

Suppose that you issue the following SELECT statement:

```
SELECT * FROM T
  WHERE C1 BETWEEN 100 AND 199 OR
        C1 BETWEEN 500 AND 599;
```

In this case, the same index can be used more than once in a multiple index access because more than one predicate could be matching. DB2 processes the query by performing the following steps:

1. DB2 retrieves all RIDs where C1 is between 100 and 199, using index IX1.
2. DB2 retrieves all RIDs where C1 is between 500 and 599, again using IX1. The union of those lists is the final set of RIDs.
3. DB2 retrieves the qualified rows by using the final RID list.

The plan table for this example is shown in the following table.

Table 128. PLAN_TABLE output for example with union (OR) operator

TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	PREFETCH	MIXOP-SEQ
T	M	0		N	L	0
T	MX	1	IX1	Y		1
T	MX	1	IX1	Y		2
T	MU	0		N		3

Example: multiple index access for XML data

Suppose that you issue the following SELECT statement that uses indexes to evaluate the XMLEXISTS predicates.

```
SELECT * FROM T
  WHERE (C1 = 1 OR C2 = 1) AND
   XMLEXISTS('/a/b[c = 1]' PASSING XML_COL1) AND
   XMLEXISTS('/a/b[(e = 2 or f[g = 3]) and h/i[j = 4] ]'

  PASSING XML_COL2);
```

The following statement shows the indexes defined on T:

```
IX1: C1
IX2: C2
VIX1: /a/b/c
VIX2: /a/b/e
VIX3: /a/b/f/g
VIX4: /a/b/h/i/j
DOCID index on T: DIX1
```

The XPath expression in the second XMLEXISTS predicate is decomposed into multiple XPath segments which are combined by AND and OR operations. DB2 matches each XPath segment to an XML index. The matching information between the predicates and indexes are as follows:

Table 129. Matching between predicates and indexes for the XMLEXISTS example

Predicate	Matching Index	XML index
C1 = 1	IX1	N

Table 129. Matching between predicates and indexes for the XMLEXISTS example (continued)

Predicate	Matching Index	XML index
C2 = 1	IX2	N
XMLEXISTS 1: /a/b[c =1]	VIX1	Y
XMLEXISTS 2: /a/b[e = 2]	VIX2	Y
XMLEXISTS 2: /a/b/f[g = 3]	VIX3	Y
XMLEXISTS2: /a/b/h/i[j = 4]	VIX4	Y

DB2 uses the above indexes to access the table T and processes the query by performing the following steps:

1. DB2 retrieves all the qualifying record identifiers (RIDs) where C1=1, by using index IX1.
2. DB2 retrieves all the qualifying RIDs where C2 = 1, by using index IX2.
3. The union of the RID lists from step 1 and 2 is the final set of qualifying RIDs where C1 = 1 OR C2 = 1.
4. DB2 retrieves all the qualifying DOCIDs where /a/b[c =1], by using index VIX1.
5. DB2 retrieves all the qualifying RIDs of the DOCID list from step 4, by using index DIX1.
6. The intersection of the RID lists from step 3 and 5 is the final set of qualifying RIDs where (C1 = 1 OR C2 = 1) AND XPath expression /a/b[c =1].
7. DB2 retrieves all the qualifying DOCIDs where /a/b[e = 2], by using index VIX2.
8. DB2 retrieves all the qualifying DOCIDs where /a/b/f[g = 3], by using index VIX3.
9. The union of the DOCID lists from step 7 and 8 is the final set of qualifying DOCIDs where XPath segment /a/b[e = 2] OR XPath segment /a/b/f[g = 3].
10. DB2 retrieves all the qualifying DOCIDs where /a/b/h/i[j = 4], by using index VIX4.
11. The intersection of the DOCID lists from step 9 and 10 is the final set of qualifying DOCIDs where (XPath segment /a/b[e = 2] OR XPath segment /a/b/f[g = 3]) AND XPath segment / a/b/h/i[j = 4].
12. DB2 retrieves all the qualifying RIDs of the DOCID list from step 11, by using index DIX1.
13. The intersection of the RID lists from step 6 and 12 is the final set of qualifying RIDs where (C1 = 1 OR C2 = 1) AND XPath expression /a/b[c =1] AND ((XPath segment /a/b[e = 2] OR XPath segment /a/b/f[g = 3]) AND XPath segment /a/b/h/i[j = 4]). DB2 accesses the data pages that are needed to retrieve the qualified rows by using the final RID list.

The plan table for this example is shown in the following table:

Table 130. The PLAN_TABLE output for the XMLEXISTS predicate example

QUERY NO	Q BLOCK NO	PLAN NO	T NAME	TAB NO	METHOD	ACCT TYPE	MATCH COLS	ACC NAME	INDEX-ONLY	MIX OP SEQ
1	1	1	T	1	0	M	0		N	0
1	1	1	T	1	0	MX	1	IX1	Y	1
1	1	1	T	1	0	MX	1	IX2	Y	2
1	1	1	T	1	0	MU	0		N	3
1	1	1	T	1	0	DX	1	VIX1	Y	4
1	1	1	T	1	0	MX	0	DIX1	Y	5
1	1	1	T	1	0	MI	0		N	6
1	1	1	T	1	0	DX	1	VIX2	Y	7
1	1	1	T	1	0	DX	1	VIX3	Y	8
1	1	1	T	1	0	DU	0		N	9
1	1	1	T	1	0	DX	1	VIX4	Y	10
1	1	1	T	1	0	DI	0		N	11
1	1	1	T	1	0	MX	1	DIX1	Y	12
1	1	1	T	1	0	MI	0		N	13

PSPI

Related concepts:

Predicates and access path selection

Access methods with XML indexes (DB2 Programming for XML)

XML data indexing (DB2 Programming for XML)

Related reference:

PLAN_TABLE

One-fetch access (ACCESSTYPE='I1')

One-fetch index access requires retrieving only one row. It is the best possible access path and is chosen whenever it is available.

PSPI

One-fetch index access applies only to statements with MIN or MAX aggregate functions: the order of the index allows a single row to give the result of the function.

One-fetch index access is a possible access path when:

- The query includes only one table.
- The query includes only one aggregate function (either MIN or MAX).
- Either no predicate or all predicates are matching predicates for the index.
- The query includes no GROUP BY clause.
- Aggregate functions are on:
  - The first index column if no predicates exist

- The last matching column of the index if the last matching predicate is a range type
- The next index column (after the last matching column) if all matching predicates are equal type

Example queries that use one-fetch index scan

Assuming that an index exists on T(C1,C2,C3), each of the following queries use one-fetch index scan:

```
SELECT MIN(C1) FROM T;
SELECT MIN(C1) FROM T WHERE C1>5;
SELECT MIN(C1) FROM T WHERE C1>5 AND C1<10;
SELECT MIN(C2) FROM T WHERE C1=5;
SELECT MAX(C1) FROM T;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2 BETWEEN 5 AND 10;
```

PSPI

Related concepts:

Calculation of aggregate values (Introduction to DB2 for z/OS)

Related reference:

MAX (DB2 SQL)

MIN (DB2 SQL)

PLAN_TABLE

Index-only access (INDEXONLY='Y')

DB2 uses *index-only access* when it can avoid accessing data pages because the information in the index satisfies the query. Conversely, when a query requests a column that is not in an index, DB2 must access the associated data pages. Because the index is almost always smaller than the table itself, index-only access paths usually process data more efficiently.

PSPI

For SELECT statements DB2 uses index-only access when all columns for a query are found in an index. For UPDATE and DELETE statements, DB2 can use index only access to qualify the selected rows. However, DB2 must access the data pages to modify the data values.

With an index on T(C1, C2), the following queries can use index-only access:

```
SELECT C1, C2 FROM T WHERE C1 > 0;
SELECT C1, C2 FROM T;
SELECT COUNT(*) FROM T WHERE C1 = 1;
```

If you create or alter a unique index that is used for index-only access, you can include extra columns in the index. By including the extra columns, you can avoid creating more indexes for index-only access on the extra columns. You specify the INCLUDE clause in a CREATE INDEX statement to add the extra columns. You can also specify ADD INCLUDE COLUMN in an ALTER INDEX statement to append a column to an existing unique index.

Index-only access to data is not possible for any step that uses list prefetch. Index-only access is not possible for padded indexes when varying-length data is returned or a VARCHAR column has a LIKE predicate. Index-only access is always possible for non-padded indexes.

If access is by more than one index, INDEXONLY is Y for a step with access type MX, or DX. The data pages are not accessed until all the steps for intersection (MI or DI) or union (MU or DU) take place.

When an SQL application uses index-only access for a rowid column, the application claims the table space or table space partition. As a result, contention might occur between the SQL application and a utility that drains the table space or partition. Index-only access to a table for a rowid column is not possible if the associated table space or partition is in an incompatible restrictive state. For example, an SQL application can make a read claim on the table space only if the restrictive state allows readers.

PSPI

Related concepts:

Multiple index access (ACCESSTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU')

Additional non-key columns in a unique index (DB2 for z/OS What's New?)

Related tasks:

Adding a column to an index when you add the column to a table (DB2 Administration Guide)

Related reference:

PLAN_TABLE

Summary of predicate processing

Equal unique index (MATCHCOLS=*number of index columns*)

An index that is fully matched and unique, and in which all matching predicates are equal-predicates, is called an *equal unique index*.

Introductory concepts

Creation of indexes (Introduction to DB2 for z/OS)

Unique indexes (Introduction to DB2 for z/OS)

PSPI

This case guarantees that only one row is retrieved. If one-fetch index access is unavailable, this is considered the most efficient access over all other indexes that are not equal unique. (The uniqueness of an index is determined by whether or not it was defined as unique.)

Sometimes DB2 can determine that an index that is not fully matching is actually an equal unique index case. Assume the following case:

```
Unique Index1: (C1, C2)
Unique Index2: (C2, C1, C3)

SELECT C3 FROM T
  WHERE C1 = 1 AND C2 = 5;
```

Index1 is a fully matching equal unique index. However, Index2 is also an equal unique index even though it is not fully matching. Index2 is the better choice because, in addition to being equal and unique, it also provides index-only access.

PSPI

Related concepts:

Index-only access (INDEXONLY='Y')

Indexes on table columns (DB2 Administration Guide)

Related reference:

PLAN_TABLE

Index access for MERGE

DB2 can always use an index for a MERGE operation when no index key columns are updated. DB2 can sometimes use an index when key columns are updated, but only when certain conditions are met.

Introductory concepts

Merge statements (Introduction to DB2 for z/OS)

Creation of indexes (Introduction to DB2 for z/OS)

The following conditions must be met to enable DB2 to use an index for a MERGE operation when index key columns are being updated:

- The MERGE statement contains a corresponding predicate in one of the following forms, for each updated index key column:
  - `*index-key-column* = constant`
  - `*index-key-column* IS NULL`
- If a view is involved, WITH CHECK OPTION is not specified.

Examples for when DB2 can and cannot use index access for MERGE statements

DB2 can use index access when the index includes no columns that are being updated

For example, assume that Index I1 on table T1 column(C1), and consider the following statement:

```
MERGE INTO T1 TRGT
  USING (VALUES (?, ?)
    FOR ? ROWS)
    AS SRC (C1,C2)
  ON (TRGT.C1 = SRC.C1)
  WHEN MATCHED THEN UPDATE SET TRGT.C2 = SRC.C2
  WHEN NOT MATCHED THEN INSERT (C1,C2)
     VALUES (SRC.C1, SRC.C2)
  NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

DB2 can use the index because the statement includes the `TRGT.C1 = SRC.C1` predicate in the ON clause and TRGT.C1 is not an updated column.

DB2 cannot use index access when the index includes columns that are being updated

For example, assume that an index I2 exists on table columns (C1, C2) and consider the following statement:

```
MERGE INTO T1 TRGT
USING (VALUES (?, ?, ?)
       FOR ? ROWS) AS SRC (C1,C2,C3)
ON TRGT.C1 = SRC.C1
AND TRGT.C2 = SRC.C2
```

```
WHEN MATCHED THEN UPDATE
SET TRGT.C1 = SRC.C1
   ,TRGT.C2 = SRC.C2
   ,TRGT.C3 = SRC.C3
WHEN NOT MATCHED THEN INSERT (C1,C2.C3)
     VALUES (SRC.C1, SRC.C2, SRC.C3)
  NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

In this case, the ON clause predicates mean that the values of C1 and C2 must already contain the values that they are updated to by the SET clause. Nevertheless, because the SET clause contains these columns, the use of I2 for index access is prevented. You can remove C1 and C2 from the SET clause to enable DB2 to use the index.

For another example, assume that an index I1 exists on columns (C1,C2,C3), and consider the following statement:

```
MERGE INTO T1 TRGT
USING (VALUES (?, ?, ?, ?)
       FOR ? ROWS) AS SRC (C1,C2,C3,C4)
ON TRGT.C1 = SRC.C1
AND (TRGT.C2 = SRC.C2 OR TRGT.C3 = SRC.C3)
WHEN MATCHED THEN UPDATE
SET TRGT.C2 = SRC.C2
   ,TRGT.C3 = SRC.C3
   ,TRGT.C4 = SRC.C4
WHEN NOT MATCHED THEN INSERT (C1,C2.C3,C4)
     VALUES (SRC.C1, SRC.C2, SRC.C3, SRC.C4)
  NOT ATOMIC CONTINUE ON SQLEXCEPTION;
```

In this case, columns C2 and C3 are included in the index that is being updated, and they are also included in index I1. DB2 is prevented from using index I1 when checking for matching rows.

The following alternative approaches can enable index access in this case:

- Create an index on column C1 only.
- Use UPDATE and INSERT statements in the application instead of MERGE. DB2 supports index access for UPDATE by using the I1 index for list prefetch when searching for rows to UPDATE. List prefetch is not supported for MERGE statements.

Related reference:

MERGE (DB2 SQL)

Index access for UPDATE

DB2 can always use an index for a UPDATE operation when no index key columns are updated. DB2 can sometimes use an index when key columns are updated, but only when certain conditions are met.

Introductory concepts

Update statements (Introduction to DB2 for z/OS)

Creation of indexes (Introduction to DB2 for z/OS)

The following conditions must be met to enable DB2 to use an index for an UPDATE operation when index key columns are being updated:

- The UPDATE statement contains a corresponding predicate in one of the following forms, for each updated index key column:
  - *index-key-column* = constant
  - *index-key-column* IS NULL

- If a view is involved, WITH CHECK OPTION is not specified.

Related reference:

UPDATE (DB2 SQL)

Range-list index scan (ACCESSTYPE='NR')

Range-list index scans are a method for simplifying the processing of OR predicates that can be mapped to a single index. This access type improves the performance of applications with data-dependent pagination.

When your SELECT statement contains an OR predicate, DB2 can use a range-list index scan and avoid scanning the index multiple times. A single index scan consumes fewer RID list resources and reduces CPU overhead than multiple index scans.

DB2 can use a range-list index scan when the SELECT statement meets the following requirements:

- Every OR predicate refers to the same table.
- Every OR predicate has at least one matching predicate.
- Every OR predicate is mapped to the same index.

Exception: DB2 does not support range-list index scans for statements that use rowset cursors.

The following SELECT statement can take benefit from range-list index scan access:

```
SELECT * FROM EMP
WHERE (LASTNAME='JONES' AND FIRSTNAME='WENDY') OR
(LASTNAME='SMITH' AND FIRSTNAME='JOHN');
```

IN-list access (ACCESSTYPE='N' or 'IN')

DB2 might choose a type of IN-list access for queries that contain IN predicates.

Introductory concepts

Values in a list (Introduction to DB2 for z/OS)

PSPI

DB2 might select either IN-list index scan or IN-list direct table access. DB2 selects which type to use based on the estimated cost of each. DB2 can also use IN-list direct table access for queries that contain more the one IN predicate.

IN-list index scan (ACCESSTYPE='N')

An *IN-list index scan* is a special case of the matching index scan, in which a single indexable IN predicate is used as a matching equal predicate. You can regard the IN-list index scan as a series of matching index scans with the values in the IN predicate being used for each matching index scan.

The following example has an index on (C1,C2,C3,C4) and might use an IN-list index scan:

```
SELECT * FROM T
  WHERE C1=1 AND C2 IN (1,2,3)
    AND C3>0 AND C4<100;
```

The plan table shows MATCHCOLS = 3 and ACCESSTYPE = N. The IN-list scan is performed as the following three matching index scans:

```
(C1=1,C2=1,C3>0), (C1=1,C2=2,C3>0), (C1=1,C2=3,C3>0)
```

Parallelism is supported for queries that involve IN-list index access, regardless of whether the IN-list access is for the inner or outer table of the parallel group.

IN-list direct table access (ACCESSTYPE='IN')

IN-list direct table access occurs when DB2 uses in-memory tables to process one or more IN-list predicates as a matching predicates. This access type is indicated in the PLAN_TABLE by ACCESSTYPE='IN'. DB2 supports matching on multiple IN-list predicates if indexes exist on the necessary columns.

For example, DB2 is likely to select IN-list direct table access for the following query:

```
SELECT *
FROM T1
WHERE T1.C1 IN ('A', 'B', 'C') AND T1.C2 IN (1, 2, 3);
```

The PLAN_TABLE, contains one row for each IN-list predicate that DB2 processes through in memory tables. For example, the plan table contains the following rows for the above example query:

Table 131. PLAN_TABLE output for IN-list direct table access

QBLOCK NO	PLAN NO	METHOD	TNAME	ACCESS TYPE	MATCH COLS	ACCESS NAME	QBLOCK _TYPE	TABLE _TYPE
1	1	0	DSNIN002(01)	IN	0		SELECT	I
1	2	1	DSNIN003(01)	IN	0		SELECT	I
1	2	1	TI	I	1	T1_IX_C1	SELECT	T

If the index is selected to access the base table for T1 and T1.C1 IN (A, B, C) is the matching predicate, the PLAN_TABLE shows two rows.

The join type between the in-memory tables is a nested-loop join. Because the access sequence for multiple IN-list predicates is the same as the index key order,DB2 avoids random I/O to access the index as much as possible. When multiple IN-list predicates are applicable to an index, DB2 decides the optimal number of matching predicates.

The first two rows show access to the in-memory tables, and the second row shows access to the base table. Nested-loop joins are used to join the tables.

PSPI

Related concepts:

How DB2 modifies IN predicates

Related reference:

PLAN_TABLE

IN predicate (DB2 SQL)

Direct row access (PRIMARY_ACCESSTYPE='D')

If an application selects a row from a table that contains a ROWID column, the row ID value implicitly contains the location of the row. If you use that row ID value in the search condition of subsequent SELECT, DELETE, or UPDATE operations, DB2 might be able to use *direct row access* navigate directly to the row.

Introductory concepts

ROWID data type (Introduction to DB2 for z/OS)

PSPI

Direct row access is very fast because DB2 does not need to use the index or a table space scan to find the row. Direct row access can be used on any table that has a ROWID column.

To use direct row access, you first select the values of a row into host variables. The value that is selected from the ROWID column contains the location of that row. Later, when you perform queries that access that row, you include the row ID value in the search condition. If DB2 determines that it can use direct row access, it uses the row ID value to navigate directly to the row.

If an application selects RID built-in function from a table, the result contains the location of the row. If you use the RID built-in function in the search condition of subsequent SELECT, DELETE, or UPDATE statements, DB2 might be able to choose direct row access to navigate directly to the row. For a query to qualify for direct row access, the search condition must be a Boolean term stage 1 predicate that fits one of these descriptions: If DB2 cannot locate the row through direct row access, it does not switch to another access method and just returns no row found. The EXPLAIN output in the PLAN_TABLE is changed if DB2 chooses the direct row access when the above conditions are satisfied, with the RID built-in function used as a search condition.

PSPI

Related concepts:

Row ID values (DB2 SQL)

Related tasks:

Specifying direct row access by using row IDs (DB2 Application programming and SQL)

Related reference:

RID (DB2 SQL)

Predicates that qualify for direct row access

For a query to qualify for direct row access, the search condition must be a Boolean term stage 1 predicate, and must meet certain conditions.

PSPI

SQL queries that meet the following criteria qualify for direct row access.

- A simple Boolean term predicate of the form COL=*noncolumn expression*, where COL has the ROWID data type and *noncolumn expression* contains a row ID

- A simple Boolean term predicate of the form COL IN *list*, where COL has the ROWID data type and the values in *list* are row IDs, and an index is defined on COL
- A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 1 or 2
- A simple Boolean term predicate of the form RID (table designator) = *noncolumn expression*, where *noncolumn expression* contains a result of the RID built-in function.
- A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 4.

However, just because a query qualifies for direct row access does not mean that access path is always chosen. If DB2 determines that another access path is better, direct row access is not chosen.

Examples

In the following predicate example, ID is a ROWID column in table T1. A unique index exists on that ID column. The host variables are of the ROWID type.

```
WHERE ID IN (:hv_rowid1,:hv_rowid2,:hv_rowid3)
```

The following predicate also qualifies for direct row access:

```
WHERE ID = ROWID(X'F0DFD230E3C0D80D81C201AA0A28010000000000000203')
```

Searching for propagated rows

If rows are propagated from one table to another, do not expect to use the same row ID value from the source table to search for the same row in the target table, or vice versa. This does not work when direct row access is the access path chosen.

Assume that the host variable in the following statement contains a row ID from SOURCE:

```
SELECT * FROM TARGET
  WHERE ID = :hv_rowid
```

Because the row ID location is not the same as in the source table, DB2 probably cannot find that row. Search on another column to retrieve the row you want.

PSPI

Reverting to ACCESSTYPE

Although DB2 might plan to use direct row access, circumstances can cause DB2 to not use direct row access at run time.

PSPI

DB2 remembers the location of the row as of the time it is accessed. However, that row can change locations (such as after a REORG) between the first and second time it is accessed, which means that DB2 cannot use direct row access to find the row on the second access attempt. Instead of using direct row access, DB2 uses the access path that is shown in the ACCESSTYPE column of PLAN_TABLE.

If the predicate you are using to do direct row access is not indexable and if DB2 is unable to use direct row access, then DB2 uses a table space scan to find the row.

This can have a profound impact on the performance of applications that rely on direct row access. Write your applications to handle the possibility that direct row access might not be used. Some options are to:

- Ensure that your application does not try to remember ROWID columns across reorganizations of the table space.

  When your application commits, it releases its claim on the table space; it is possible that a REORG can run and move the row, which disables direct row access. Plan your commit processing accordingly; use the returned row ID value before committing, or re-select the row ID value after a commit is issued.

  If you are storing ROWID columns from another table, update those values after the table with the ROWID column is reorganized.
- Create an index on the ROWID column, so that DB2 can use the index if direct row access is disabled.
- Supplement the ROWID column predicate with another predicate that enables DB2 to use an existing index on the table. For example, after reading a row, an application might perform the following update:

  ```
  EXEC SQL UPDATE EMP
  SET SALARY = :hv_salary + 1200
    WHERE EMP_ROWID = :hv_emp_rowid
    AND EMPNO = :hv_empno;
  ```

  If an index exists on EMPNO, DB2 can use index access if direct access fails. The additional predicate ensures DB2 does not revert to a table space scan.

PSPI

Access methods that prevent direct row access

Certain access methods prevent DB2 from using direct row access.

Parallelism

PSPI

Direct row access and parallelism are mutually exclusive. If a query qualifies for both direct row access and parallelism, direct row access is used. If direct row access fails, DB2 does not revert to parallelism; instead it reverts to the backup access type (as designated by column ACCESSTYPE in the PLAN_TABLE). This might result in a table space scan. To avoid a table space scan in case direct row access fails, add an indexed column to the predicate.

RID list processing

Direct row access and RID list processing are mutually exclusive. If a query qualifies for both direct row access and RID list processing, direct row access is used. If direct row access fails, DB2 does not revert to RID list processing; instead it reverts to the backup access type.

PSPI

Example: Coding with row IDs for direct row access

You can obtain the row ID value for a row, and then to use that value to find the row efficiently when you want to modify it.

PSPI

The following figure is a portion of a C program that shows you how to obtain the row ID value for a row and use that value to find the row efficiently.

```
/**************************/
/* Declare host variables */
/**************************/
EXEC SQL BEGIN DECLARE SECTION;
   SQL TYPE IS BLOB_LOCATOR hv_picture;
   SQL TYPE IS CLOB_LOCATOR hv_resume;
   SQL TYPE IS ROWID hv_emp_rowid;
   short hv_dept, hv_id;
   char hv_name[30];
   decimal hv_salary[5,2];
EXEC SQL END DECLARE SECTION;

/**********************************************************/
/* Retrieve the picture and resume from the PIC_RES table */
/**********************************************************/
strcpy(hv_name, "Jones");
EXEC SQL SELECT PR.PICTURE, PR.RESUME INTO :hv_picture, :hv_resume
  FROM PIC_RES PR
  WHERE PR.Name = :hv_name;

/**********************************************************/
/* Insert a row into the EMPDATA table that contains the  */
/* picture and resume you obtained from the PIC_RES table */
/**********************************************************/
EXEC SQL INSERT INTO EMPDATA
  VALUES (DEFAULT,9999,'Jones', 35000.00, 99,
  :hv_picture, :hv_resume);

/**********************************************************/
/* Now retrieve some information about that row,          */
/* including the ROWID value.                             */
/**********************************************************/
hv_dept = 99;
EXEC SQL SELECT E.SALARY,  E.EMP_ROWID
  INTO :hv_salary, :hv_emp_rowid
  FROM EMPDATA E
  WHERE E.DEPTNUM = :hv_dept AND E.NAME = :hv_name;
/**********************************************************/
/* Update columns SALARY, PICTURE, and RESUME.  Use the   */
/* ROWID value you obtained in the previous statement     */
/* to access the row you want to update.                  */
/* smiley_face and update_resume are                      */
/* user-defined functions that are not shown here.        */
/**********************************************************/
EXEC SQL UPDATE EMPDATA
  SET SALARY = :hv_salary + 1200,
  PICTURE = smiley_face(:hv_picture),
  RESUME = update_resume(:hv_resume)
  WHERE EMP_ROWID = :hv_emp_rowid;

/**********************************************************/
/* Use the ROWID value to obtain the employee ID from the */
/* same record.                                           */
/**********************************************************/
EXEC SQL SELECT E.ID INTO :hv_id
  FROM EMPDATA E
  WHERE E.EMP_ROWID = :hv_emp_rowid;

/**********************************************************/
/* Use the ROWID value to delete the employee record      */
/* from the table.                                         */
/**********************************************************/
EXEC SQL DELETE FROM EMPDATA
  WHERE EMP_ROWID = :hv_emp_rowid;
```

Figure 50. Example of using a row ID value for direct row access

PSPI

Page range screening (PAGE_RANGE='Y')

DB2 can use *page range screening* to limit a scan of data in a partitioned table space to one or more partitions. This access method is also sometimes called a *limited partition scan*.

PSPI

Subject to certain exceptions, a predicate on any column of the partitioning key might be used for page range screening if that predicate can eliminate partitions from the scan.

Page range screening can be combined with other access methods. For example, consider the following query:

```
SELECT .. FROM T
   WHERE (C1 BETWEEN '2002' AND '3280'
   OR C1 BETWEEN '6000' AND '8000')
   AND C2 = '6';
```

Assume that table T has a partitioned index on column C1 and that values of C1 between 2002 and 3280 all appear in partitions 3 and 4 and the values between 6000 and 8000 appear in partitions 8 and 9. Assume also that T has another index on column C2. DB2 could choose any of these access methods:

- A matching index scan on column C1. The scan reads index values and data only from partitions 3, 4, 8, and 9. (PAGE_RANGE='N')
- A matching index scan on column C2. (DB2 might choose that if few rows have C2=6.) The matching index scan reads all RIDs for C2=6 from the index on C2 and corresponding data pages from partitions 3, 4, 8, and 9. (PAGE_RANGE='Y')
- A table space scan on T. DB2 avoids reading data pages from any partitions except 3, 4, 8 and 9. (PAGE_RANGE='Y').

PSPI

Related concepts:

Efficient queries for tables with data-partitioned secondary indexes

Parallel processing access (PARALLELISM_MODE='I', 'C', or 'X')

DB2 can use parallel processing for read-only queries.

PSPI

If the PARALLELISM_MODE value is:
DB2 plans to use:

I Parallel I/O operations. Query I/O parallelism is deprecated and is likely to be removed in a future release.

C Parallel CP operations.

X Sysplex query parallelism. Sysplex query parallelism is deprecated and is likely to be removed in a future release.

Non-null values in the ACCESS_DEGREE and JOIN_DEGREE columns indicate the degree of parallelism that DB2 plans to use. However, at execution time, DB2 might not actually use parallelism, or use fewer operations in parallel than originally planned.

PSPI

Related tasks:

Interpreting query parallelism

Programming for parallel processing

Tuning parallel processing

Disabling query parallelism

Complex trigger WHEN clause access (QBLOCKTYPE='TRIGGR')

The plan table does not report simple trigger WHEN clauses, such as WHEN (N.C1 < 5). However, the plan table does report complex trigger WHEN clauses, which are clauses that involve other base tables and transition tables.

PSPI

The QBLOCK_TYPE column of the top level query block shows TRIGGR to indicate a complex trigger WHEN clause.

Consider the following trigger:

```
CREATE TRIGGER REORDER
       AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
       REFERENCING NEW TABLE AS NT OLD AS O
       FOR EACH STATEMENT MODE DB2SQL
       WHEN (O.ON_HAND < (SELECT MAX(ON_HAND) FROM NT))
          BEGIN ATOMIC
              INSERT INTO ORDER_LOG VALUES (O.PARTNO, O.ON_HAND);
          END
```

The following table shows the corresponding plan table for the WHEN clause.

Table 132. Plan table for the WHEN clause

QBLOCKNO	PLANNO	TABLE	ACCESSTYPE	QBLOCK_TYPE	PARENT_QBLOCKNO
1	1			TRIGGR	0
2	1	NT	R	NCOSUB	1

PSPI

Related concepts:

Trigger cascading (DB2 Application programming and SQL)

Interactions between triggers and referential constraints (DB2 Application programming and SQL)

Related tasks:

Creating a trigger (DB2 Application programming and SQL)

Related reference:

CREATE TRIGGER (DB2 SQL)

Prefetch access paths (PREFETCH='D', 'S', 'L', or 'U')

Prefetch is a mechanism for reading a set of pages into the buffer pool with only one asynchronous I/O operation.

Prefetch can allow substantial savings in both CPU and I/O by avoiding costly synchronous read I/O operations. To achieve those savings, you can monitor the use of prefetch, and take steps to enable your queries to take advantage of optimal prefetch access paths.

DB2 uses a process called sequential detection to determine whether to use prefetch and the type of prefetch to use. DB2 uses different types of prefetch in different situations, including dynamic prefetch, sequential prefetch, and list prefetch.

Related concepts:

Read operations and prefetch I/O

Dynamic prefetch (PREFETCH='D')

With *dynamic prefetch*, DB2 uses a sequential detection algorithm to determine whether data pages are being read sequentially.

PSPI

Dynamic prefetch offers improved performance over sequential prefetch, especially when DB2 cannot detect whether the pages can be accessed sequentially (because the catalog statistics might not always be correct). When DB2 chooses dynamic prefetch, the value of the PREFETCH column is normally set to 'D.' However, DB2 might also use dynamic prefetch when the value of PREFETCH indicates 'S.'

When dynamic prefetch is used

DB2 uses dynamic prefetch to read data pages whenever an index is used to determine which data pages contain the required rows.

DB2 also uses dynamic prefetch to avoid synchronous I/O for index pages when it scans the leaf pages an index in key-sequential order. An organized index is likely to trigger dynamic prefetch I/Os. However, when the leaf pages of an index are not well organized, dynamic prefetch is not triggered, and some synchronous I/O might occur. However, when DB2 detects the synchronous I/OS it can switch to list prefetch for the next set of leaf pages. In that case, synchronous I/O occurs only for the non-leaf pages.

PSPI

Related concepts:

Read operations and prefetch I/O

Sequential prefetch (PREFETCH='S')

Sequential prefetch is used for table scans. The maximum number of pages read by a request issued from your application program is determined by the size of the buffer pool used.

PSPI

For certain utilities (LOAD, REORG, RECOVER), the prefetch quantity can be twice as much.

When sequential prefetch is used

Sequential prefetch is used only for table space scans. When DB2 expects that sequential prefetch will probably be used, the value of the PREFETCH column is 'S.' However, DB2 might use dynamic prefetch in some cases if when the value of PREFETCH column is 'S.'

PSPI

Related concepts:

Read operations and prefetch I/O

List prefetch (PREFETCH='L' or 'U')

List prefetch reads a set of data pages determined by a list of record identifiers (RIDs) taken from an index.

PSPI

List prefetch access paths are ideally suited for queries where the qualified rows, as determined in index key sequence, are not sequential, are skip-sequential but sparse, or when the value of the DATAREPEATFACTORF statistic is large.

With list prefetch the fetched data pages do not need to be contiguous. The maximum number of pages that can be retrieved in a single list prefetch is 32 (64 for utilities). The value of the PREFETCH column is set to 'L' or 'U' when DB2 expects to use list prefetch.

List prefetch can be used with either single or multiple index access.

When the optimizer chooses the list prefetch access path, DB2 uses the following process:

1. Retrieve the list of rows identifiers through a matching index scan on one or more index.
2. Sort the list of row identifiers in ascending order by page number.
3. Prefetch the pages in order, by using the sorted list of row identifiers.

List prefetch does not preserve the data ordering given by the index. Because the RIDs are sorted in page number order before data access, the data is not retrieved in order by any column. If the data must be ordered for an ORDER BY clause or any other reason, it requires an additional sort. DB2 sometimes uses list prefetch without sorting the RID list for performance reasons. When that happens, the value of the PREFETCH column is set to 'U'.

In a hybrid join, if the index is highly clustered, the page numbers might not be sorted before accessing the data.

List prefetch can be used with most matching predicates for an index scan.

When list prefetch is used

List prefetch is used for the following operations:

- Typically with a single index that has a cluster ratio lower than 80%
- Sometimes on indexes with a high cluster ratio, if the estimated amount of data to be accessed is too small to make sequential prefetch efficient, but large enough to require more than one regular read
- Always to access data by multiple index access
- Always to access data from the inner table during a hybrid join
- Typically for updatable cursors when the index contains columns that might be updated.
- When IN-list predicates are used through an in-memory table as matching predicates (ACCESSTYPE='IN').

DB2 uses the RID pool for list prefetch processing. The MAXRBLK subsystem parameter controls the maximum size of the RID pool. If a single list prefetch operation tries to use too much of the RID pool or attempts to read too many rows from the table, the access path might revert to a table space scan. However, you can specify that RID list processing continues in work files by setting the value of the MAXTEMPS_RID subsystem parameter.

Advantages of list prefetch

List prefetch is most useful for *skip-sequential access* when a number of non-sequential data pages are accessed in sequential order, but with intervening pages that do not contain needed data. In such cases, dynamic prefetch reads all of the intervening pages, the number of asynchronous pages read exceeds the number of get page operations, and the buffer pool is not used efficiently. List prefetch offers the following specific advantages over dynamic prefetch:

- List prefetch uses buffers very economically.
- List prefetch is not sensitive to index cluster ratio and performs much better than dynamic prefetch when the data getpages are sparse.
- Sorted list prefetch never uses two getpage operations for the same data page.
- If several data pages need to be skipped, list prefetch minimizes the channel time, enabling faster I/O than dynamic prefetch if the control unit hit ration is high.
- For some types of control units, list prefetch is faster than sequential I/O for skip-sequential access. You can check with your storage vendor to learn whether that is true for your type of control unit.

Disadvantages of list prefetch

However, when compared to dynamic prefetch, list prefetch also has certain disadvantages :

- Dynamic prefetch outperforms list prefetch when row identifiers are very dense, such as range scans when the cluster ratio is high.
- For some types of control units, list prefetch is slower than sequential I/O for skip-sequential access. You can check with your storage vendor to learn whether that is true for your type of control unit.

- The SEQCACH installation parameter does not apply to list prefetch. Therefore, data might not be streamed into the control unit cache from the disk on some control units models. This lack of streaming might or might not hurt performance because some types of control units might prefer to target the prefetch to a smaller subset of pages. You can check with your storage vendor to find out how your type of control unit performs in these situations.
- For queries that use an ORDER BY or GROUP BY clause that uses an index key column, list prefetch always requires a be sort of the result set, whereas dynamic prefetch does not always require a sort. The cost of the sort depends on of the size of the result set, rather than the number of data pages read.
- If an application prematurely closes a cursor before fetching the entire result set, the time that list prefetch used to process the index and create the sorted RID list is wasted.

Related concepts:

Read operations and prefetch I/O

Additional statistics that provide index costs

Related tasks:

Managing RID pool size

Related reference:

RID POOL SIZE field (MAXRBLK subsystem parameter) (DB2 Installation and Migration)

MAX TEMP RID field (MAXTEMPS_RID subsystem parameter) (DB2 Installation and Migration)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Related information:

DB2 for z/OS and List Prefetch Optimizer (IBM Redbooks)

Sort access

DB2 can use two general types of sorts that DB2 can use when accessing data. One is a sort of data rows; the other is a sort of row identifiers (RIDs) in a RID list. You can use DB2 EXPLAIN to determine whether a SQL statement requires sort operations.

Sorts of data

After you run EXPLAIN, DB2 sorts are indicated in PLAN_TABLE. The sorts can be either sorts of the composite table or the new table.

PSPI

If a single row of PLAN_TABLE has a 'Y' in more than one of the sort composite columns, then one sort accomplishes two things. (DB2 does not perform two sorts when two 'Y's are in the same row.) For instance, if both SORTC_ORDERBY and SORTC_UNIQ are 'Y' in one row of PLAN_TABLE, then a single sort puts the rows in order and removes any duplicate rows as well.

The only reason DB2 sorts the new table is for join processing, which is indicated by SORTN_JOIN.

Sorts for GROUP BY and ORDER BY

These sorts are indicated by SORTC_ORDERBY, and SORTC_GROUPBY in PLAN_TABLE.

If the statement includes both a GROUP BY clause and an ORDER BY clause, and if every item in the ORDER-BY list is in the GROUP-BY list, then only one sort is performed, which is marked as SORTC_ORDERBY.

The performance of the sort by the GROUP BY clause is improved when the query accesses a single table and when the GROUP BY column has no index.

Sorts to remove duplicates

This type of sort is used to process a query with SELECT DISTINCT, with a set function such as COUNT(DISTINCT COL1), or to remove duplicates in UNION processing. It is indicated by SORTC_UNIQ in PLAN_TABLE.

Sorts used in join processing

For hybrid join (METHOD 4) and nested loop join (METHOD 1), the composite table can be sorted to make the join more efficient. For merge join (METHOD 2), both the composite table and new table need to be sorted unless an index is used for accessing these tables that gives the correct order already. The sorts needed for join processing are indicated by SORTN_JOIN and SORTC_JOIN in the PLAN_TABLE.

When SORTN_JOIN is set, each of the following join method behaves differently:

Nested loop join
SORTN_JOIN is only valid in support of star join. When SORTN_JOIN = Y for nested loop join, the qualified rows from the dimension or snowflake are sorted into the fact table join column sequence. A sparse index might also be created as a result of the sort to support efficient feedback loop skipping processing which is part of the star-join execution.

Sort merge join
The new table is accessed and sorted into join column sequence.

Hybrid join
When SORTN_JOIN is on, the intermediate table is sorted into inner table rid sequence to support efficient list prefetch access to the inner table data.

PSPI

Sorts for subquery processing

When a noncorrelated IN or NOT IN subquery is present in the query, the results of the subquery are sorted and put into a work file for later reference by the parent query.

The results of the subquery are sorted because this allows the parent query to be more efficient when processing the IN or NOT IN predicate. Duplicates are not needed in the work file, and are removed. Noncorrelated subqueries used with =ANY or =ALL, or NOT=ANY or NOT=ALL also need the same type of sort as IN or NOT IN subqueries. When a sort for a noncorrelated subquery is performed, you see both SORTC_ORDERBY and SORTC_UNIQUE in PLAN_TABLE. This is because DB2 removes the duplicates and performs the sort.

SORTN_GROUPBY, SORTN_ORDERBY, and SORTN_UNIQ are not currently used by DB2.

PSPI

Sorts of RIDs

To perform list prefetch, DB2 sorts RIDs into ascending page number order. This sort is very fast and is done totally in memory.

PSPI

A RID sort is usually not indicated in the PLAN_TABLE, but a RID sort normally is performed whenever list prefetch is used. The only exception to this rule is when a hybrid join is performed and a single, highly clustered index is used on the inner table. In this case SORTN_JOIN is 'N', indicating that the RID list for the inner table was not sorted.

PSPI

Related concepts:

Access methods that prevent direct row access

List prefetch (PREFETCH='L' or 'U')

Hybrid join (METHOD=4)

The effect of sorts on OPEN CURSOR

The type of sort processing required by the cursor affects the amount of time it can take for DB2 to process the OPEN CURSOR statement.

PSPI

This information outlines the effect of sorts and parallelism on OPEN CURSOR.

Without parallelism:

- If no sorts are required, then OPEN CURSOR does not access any data. It is at the first fetch that data is returned.
- If a sort is required, then the OPEN CURSOR causes the materialized result table to be produced. Control returns to the application after the result table is materialized. If a cursor that requires a sort is closed and reopened, the sort is performed again.
- If a RID sort is performed, but no data sort, then it is not until the first row is fetched that the RID list is built from the index and the first data record is returned. Subsequent fetches access the RID pool to access the next data record.

With parallelism:

- At OPEN CURSOR, parallelism is asynchronously started, regardless of whether a sort is required. Control returns to the application immediately after the parallelism work is started.
- If a RID sort is performed, but no data sort, then parallelism is not started until the first fetch. This works the same way as with no parallelism.

PSPI

Investigating join operations

A *join* operation retrieves rows from more than one table and combines them. The operation specifies at least two tables, but the two tables need not be distinct.

Composite tables

Introductory concepts

Ways to join data from more than one table (Introduction to DB2 for z/OS)

PSPI

A *composite table* represents the result of accessing one or more tables in a query. If a query contains a single table, only one composite table exists. If one or more joins are involved, an *outer composite table* consists of the intermediate result rows from the previous join step. This intermediate result might, or might not, be materialized into a work file.

The *new table* (or *inner table*) in a join operation is the table that is newly accessed in the step.

A join operation can involve more than two tables. In these cases, the operation is carried out in a series of steps. For non-star joins, each step joins only two tables.

Composite table example

The following figure shows a two-step join operation.

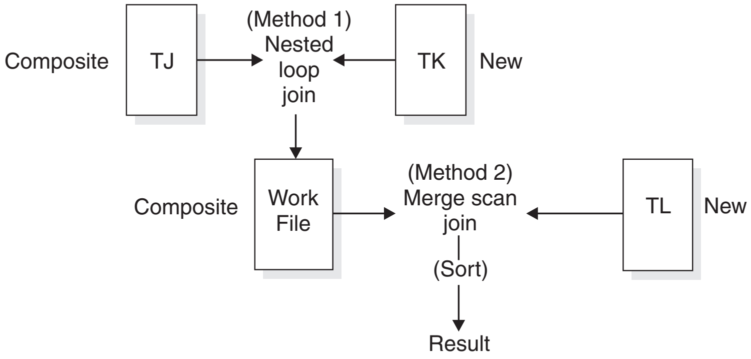


Figure 51. Two-step join operation

DB2 performs the following steps to complete the join operation:

1. Accesses the first table (METHOD=0), named TJ (TNAME), which becomes the composite table in step 2.
2. Joins the new table TK to TJ, forming a new composite table.
3. Sorts the new table TL (SORTN_JOIN=Y) and the composite table (SORTC_JOIN=Y), and then joins the two sorted tables.
4. Sorts the final composite table (TNAME is blank) into the order (SORTC_ORDERBY=Y) that you want.

The following tables show a subset of columns in a plan table for this join operation:

Table 133. Subset of columns for a two-step join operation

METHOD	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	INDEX-ONLY	TSLOCK-MODE
0	TJ	I	1	TJX1	N	IS
1	TK	I	1	TKX1	N	IS
2	TL	I	0	TLX1	Y	S
3			0		N	

Table 134. Subset of columns for a two-step join operation

SORTN UNIQ	SORTN JOIN	SORTN ORDERBY	SORTN GROUPBY	SORTC UNIQ	SORTC JOIN	SORTC ORDERBY	SORTC GROUPBY
N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N
N	Y	N	N	N	Y	N	N
N	N	N	N	N	N	Y	N

Join conditions

A join operation typically matches a row of one table with a row of another on the basis of a *join condition*. For example, the condition might specify that the value in column A of one table equals the value of column X in the other table (WHERE T1.A = T2.X).

Two kinds of joins differ in what they do with rows in one table that do not match on the join condition with any row in the other table:

- An *inner join* discards rows of either table that do not match any row of the other table.
- An *outer join* keeps unmatched rows of one or the other table, or of both. A row in the composite table that results from an unmatched row is filled out with null values. As the following table shows, outer joins are distinguished by which unmatched rows they keep.

Table 135. Join types and kept unmatched rows

Outer join type	Included unmatched rows
Left outer join	The composite (outer) table
Right outer join	The new (inner) table
Full outer join	Both tables

Join condition example

Suppose that you issue the following statement to explain an outer join:

```
EXPLAIN PLAN SET QUERYNO = 10 FOR
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
  FROM PROJECTS LEFT JOIN
```

```
     (SELECT PART,
        COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
        PRODUCTS.PRODUCT
        FROM PARTS FULL OUTER JOIN PRODUCTS
            ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
 ON PROJECTS.PROD# = PRODNUM
```

The following table shows a subset of the plan table for the outer join.

Table 136. Plan table output for an example with outer joins

QUERYNO	QBLOCKNO	PLANNO	TNAME	JOIN_TYPE
10	1	1	PROJECTS	
10	1	2	TEMP	L
10	2	1	PRODUCTS	
10	2	2	PARTS	F

Column JOIN_TYPE identifies the type of outer join with one of these values:
- F for FULL OUTER JOIN
- L for LEFT OUTER JOIN
- Blank for INNER JOIN or no join

At execution, DB2 converts every right outer join to a left outer join; thus JOIN_TYPE never identifies a right outer join specifically.

Materialization with outer join

Sometimes DB2 has to materialize a result table when an outer join is used in conjunction with other joins, views, or nested table expressions. You can tell when this happens by looking at the TABLE_TYPE and TNAME columns of the plan table. When materialization occurs, TABLE_TYPE contains a W, and TNAME shows the name of the materialized table as DSNWFQB(*xx*), where *xx* is the number of the query block (QBLOCKNO) that produced the work file.

PSPI

Related concepts:

Outer joins (DB2 Application programming and SQL)

Materialization

Related tasks:

Joining data from more than one table (DB2 Application programming and SQL)

Related reference:

joined-table (DB2 SQL)

join-condition (DB2 SQL)

Cartesian join with small tables first

A *Cartesian join* is a form of nested loop join in which no join predicates exist between the two tables.

PSPI

DB2 usually avoids a Cartesian join, but sometimes it is the most efficient method, as in the following example. The query uses three tables: T1 has 2 rows, T2 has 3 rows, and T3 has 10 million rows.

```
SELECT * FROM T1, T2, T3
  WHERE  T1.C1 = T3.C1 AND
         T2.C2 = T3.C2 AND
         T3.C3 = 5;
```

Join predicates are between T1 and T3 and between T2 and T3. No predicate joins T1 and T2.

Assume that 5 million rows of T3 have the value C3=5. Processing time is large if T3 is the outer table of the join and tables T1 and T2 are accessed for each of 5 million rows.

However if all rows from T1 and T2 are joined, without a join predicate, the 5 million rows are accessed only six times, once for each row in the Cartesian join of T1 and T2. It is difficult to say which access path is the most efficient. DB2 evaluates the different options and could decide to access the tables in the sequence T1, T2, T3.

PSPI

Nested loop join (METHOD=1)

In *nested loop join* DB2 scans the composite (outer) table. For each row in that table that qualifies (by satisfying the predicates on that table), DB2 searches for matching rows of the new (inner) table.

PSPI

DB2 concatenates any matching rows that it finds with the current row of the composite table. If no rows match the current row, then:

- For an inner join, DB2 discards the current row.
- For an outer join, DB2 concatenates a row of null values.

Stage 1 and stage 2 predicates eliminate unqualified rows during the join. (For an explanation of those types of predicate, see “Stage 1 and stage 2 predicates” on page 341.) DB2 can scan either table using any of the available access methods, including table space scan.

PSPI

Performance considerations for nested loop join

A *nested loop join* repetitively scans the inner table of the join.

That is, DB2 scans the outer table once, and scans the inner table as many times as the number of qualifying rows in the outer table. Therefore, the nested loop join is usually the most efficient join method when the values of the join column passed to the inner table are in sequence and the index on the join column of the inner table is clustered, or the number of rows retrieved in the inner table through the index is small.

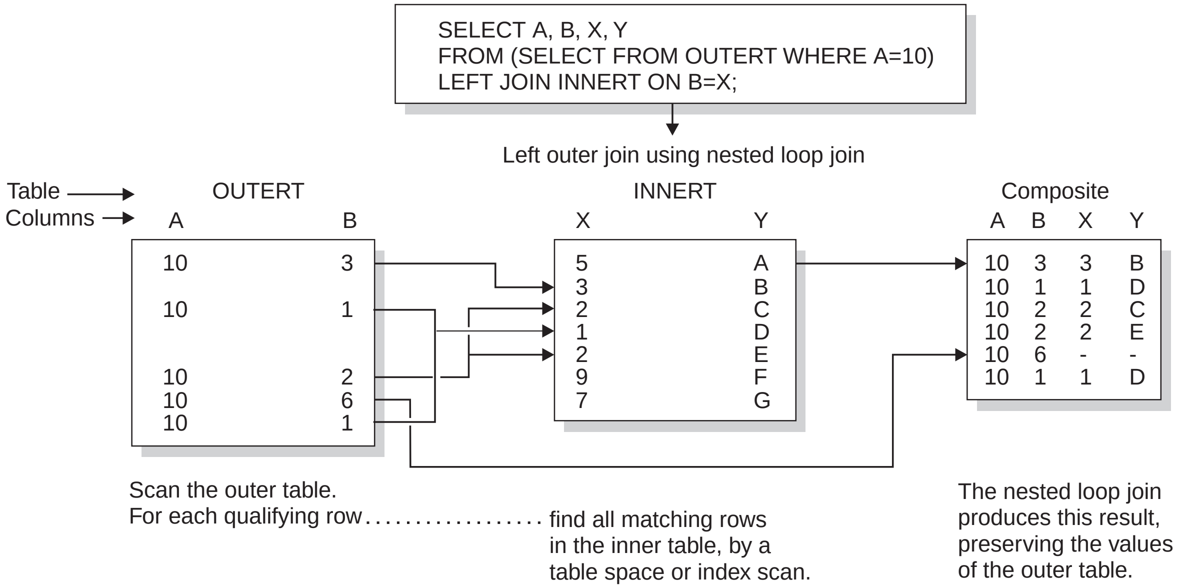
When nested loop join is used

DB2 often uses a nested loop join in the following situations.

- The outer table is small.
- Predicates with small filter factors reduce the number of qualifying rows in the outer table.
- Either an efficient, highly clustered index exists on the join columns of the inner table, or DB2 can dynamically create a sparse index on the inner table, and use that index for subsequent access.
- The number of data pages that are accessed in the inner table is small.
- No join columns exist. Hybrid and sort-merge joins require join columns; nested loop joins do not.

Example: left outer join

The following figure illustrates a nested loop for a left outer join. The outer join preserves the unmatched row in OUTERT with values A=10 and B=6. The same join method for an inner join differs only in discarding that row. The following figure illustrates a nested loop join.



```
SELECT A, B, X, Y
FROM (SELECT FROM OUTERT WHERE A=10)
LEFT JOIN INNERT ON B=X;
```

OUTERT

A	B
10	3
10	1
10	2
10	6
10	1

INNERT

X	Y
5	A
3	B
2	C
1	D
2	E
9	F
7	G

Composite

A	B	X	Y
10	3	3	B
10	1	1	D
10	2	2	C
10	2	2	E
10	6	-	-
10	1	1	D

Scan the outer table.
For each qualifying row.................. find all matching rows in the inner table, by a table space or index scan.

The nested loop join produces this result, preserving the values of the outer table.

Figure 52. Nested loop join for a left outer join

Example: one-row table priority

For a case like the following example, with a unique index on T1.C2, DB2 detects that T1 has only one row that satisfies the search condition. DB2 makes T1 the first table in a nested loop join.

```
SELECT * FROM T1, T2
  WHERE  T1.C1 = T2.C1 AND
         T1.C2 = 5;
```

Example: Cartesian join with small tables first

A *Cartesian join* is a form of nested loop join in which no join predicates exist between the two tables. DB2 usually avoids a Cartesian join, but sometimes it is the most efficient method, as in the following example. The query uses three tables: T1 has 2 rows, T2 has 3 rows, and T3 has 10 million rows.

```
SELECT * FROM T1, T2, T3
  WHERE  T1.C1 = T3.C1 AND
         T2.C2 = T3.C2 AND
         T3.C3 = 5;
```

Join predicates are between T1 and T3 and between T2 and T3. No predicate joins T1 and T2.

Assume that 5 million rows of T3 have the value C3=5. Processing time is large if T3 is the outer table of the join and tables T1 and T2 are accessed for each of 5 million rows.

However, if all rows from T1 and T2 are joined, without a join predicate, the 5 million rows are accessed only six times, once for each row in the Cartesian join of T1 and T2. It is difficult to say which access path is the most efficient. DB2 evaluates the different options and could decide to access the tables in the sequence T1, T2, T3.

PSPI

Sorts on the composite table

PSPI

DB2 might sort the composite table under the following conditions:

- The join columns in the composite table and the new table are not in the same sequence.
- The join column of the composite table has no index.
- The index is poorly clustered.

Nested loop join with a sorted composite table has the following performance advantages:

- Uses sequential detection efficiently to prefetch data pages of the new table, reducing the number of synchronous I/O operations and the elapsed time.
- Avoids repetitive full probes of the inner table index by using the index look-aside.

PSPI

Nested loop join with sparse index access

A value of PRIMARY_ACCESSTYPE = T indicates that DB2 dynamically creates a sparse index on the inner table and uses the sparse index to search the work file that is built on the inner table.

Nested loop join with sparse index has the following performance advantages:

- Access to the inner table is more efficient when the inner has no efficient index on the join columns.
- A sort of the composite table is avoided when composite table is relatively large.

Memory allocation for sparse index is controlled by the value of the MXDTCACH subsystem parameter. However, if the available storage is insufficient when the query executes, DB2 might be unable to build the sparse index, degrading performance. The solution is to resolve the memory shortage. However, if you cannot do so, reduce the MXDTCACH setting.

PSPI

Related reference:

MAX DATA CACHING field (MXDTCACH subsystem parameter) (DB2 Installation and Migration)

When a MERGE statement is used (QBLOCK_TYPE ='MERGE')

You can determine whether a MERGE statement was used and how it was processed by analyzing the QBLOCK_TYPE and PARENT_QBLOCKNO columns.

PSPI

If the QBLOCK_TYPE column contains MERGE, a MERGE statement was used. In most cases, a MERGE is processed in multiple query blocks with QBLOCK_TYPEs MERGE, UPDATE, and INSERT.

Example

Consider the following MERGE statement:

```
MERGE INTO ARCHIVE AR
  USING VALUES (:hv_activity, :hv_description) FOR
    :hv_nrows ROWS as AC (ACTIVITY, DESCRIPTION)
    ON (AR.ACTIVITY = AC.ACTIVITY)
  WHEN MATCHED THEN UPDATE SET DESCRIPTION = AC.DESCRIPTION
  WHEN NOT MATCHED THEN INSERT (ACTIVITY, DESCRIPTION)
   VALUES (AC.ACTIVITY, AC.DESCRIPTION)
   NOT ATOMIC CONTINUE ON SQL EXCEPTION
```

The following table shows the corresponding plan table for the MERGE statement.

Table 137. Plan table for MERGE statement

QBLOCK NO	PARENT_ QBLOCKNO	QBLOCK_ TYPE	PLANNO	TNAME	TABLE_ TYPE	JOIN_ TYPE	METHOD
1	0	MERGE	1	ACTIVITIES	B		
1	0	MERGE	2	ARCHIVE	T	L	1
2	1	UPDATE	1	ARCHIVE	T		
3	1	INSERT	1	ARCHIVE	T		

PSPI

Merge scan join (METHOD=2)

Merge scan join is also known as *merge join* or *sort merge join*. For this method, there must be one or more predicates of the form TABLE1.COL1=TABLE2.COL2, where the two columns have the same data type and length attribute.

PSPI

The following figure illustrates a merge scan join.

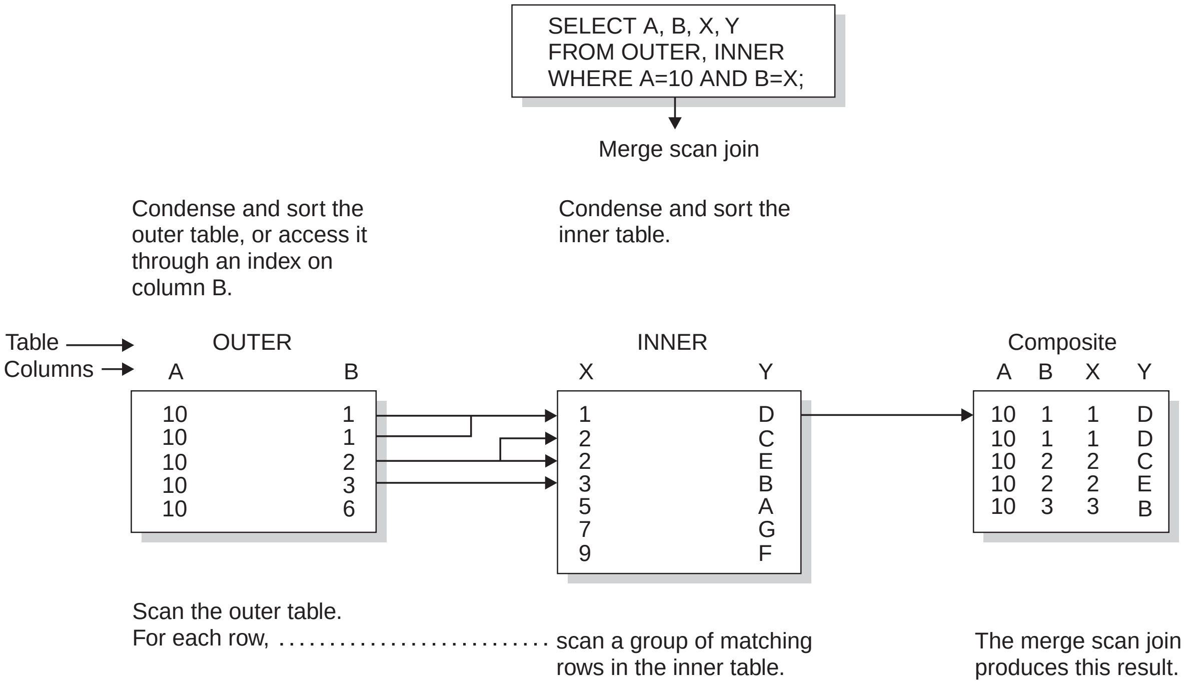


Figure 53. Merge scan join

DB2 scans both tables in the order of the join columns. If no efficient indexes on the join columns provide the order, DB2 might sort the outer table, the inner table, or both. The inner table is put into a work file; the outer table is put into a work file only if it must be sorted. When a row of the outer table matches a row of the inner table, DB2 returns the combined rows.

DB2 then reads another row of the inner table that might match the same row of the outer table and continues reading rows of the inner table as long as a match is found. When a match is no longer found, DB2 reads another row of the outer table.

- If that row has the same value in the join column, DB2 reads again the matching group of records from the inner table. Thus, a group of duplicate records in the inner table is scanned one time for each matching record in the outer table.
- If the outer row has a new value in the join column, DB2 searches ahead in the inner table. It can find any of the following rows:
  - Unmatched rows in the inner table, with lower values in the join column.
  - A new matching inner row. DB2 then starts the process again.
  - An inner row with a higher value of the join column. Now the row of the outer table is unmatched. DB2 searches ahead in the outer table, and can find any of the following rows:
    - Unmatched rows in the outer table.
    - A new matching outer row. DB2 then starts the process again.
    - An outer row with a higher value of the join column. Now the row of the inner table is unmatched, and DB2 resumes searching the inner table.

If DB2 finds an unmatched row:

For an inner join, DB2 discards the row.

For a left outer join, DB2 discards the row if it comes from the inner table and keeps it if it comes from the outer table.

For a full outer join, DB2 keeps the row.

When DB2 keeps an unmatched row from a table, it concatenates a set of null values as if that matched from the other table. A merge scan join must be used for a full outer join.

Performance considerations for merge scan join

A full outer join by this method uses all predicates in the ON clause to match the two tables and reads every row at the time of the join. Inner and left outer joins use only stage 1 predicates in the ON clause to match the tables. If your tables match on more than one column, it is generally more efficient to put all the predicates for the matches in the ON clause, rather than to leave some of them in the WHERE clause.

For an inner join, DB2 can derive extra predicates for the inner table at bind time and apply them to the sorted outer table to be used at run time. The predicates can reduce the size of the work file needed for the inner table.

If DB2 has used an efficient index on the join columns, to retrieve the rows of the inner table, those rows are already in sequence. DB2 puts the data directly into the work file without sorting the inner table, which reduces the elapsed time.

You cannot use RANDOM order index columns as part of a sort merge join. If a join is between one table with that has an ASC index on the join column and a second table that has a RANDOM index, the indexes are in completely different orders, and cannot be merged.

When merge scan join is used

- The qualifying rows of the inner and outer table are large, and the join predicate does not provide much filtering; that is, in a many-to-many join.
- The tables are large and have no indexes with matching columns.
- Few columns are selected on inner tables. This is the case when a DB2 sort is used. The fewer the columns to be sorted, the more efficient the sort is.

PSPI

Hybrid join (METHOD=4)

The hybrid join method applies only to an inner join, and requires an index on the join column of the inner table.

PSPI

The following figure illustrates a hybrid join.

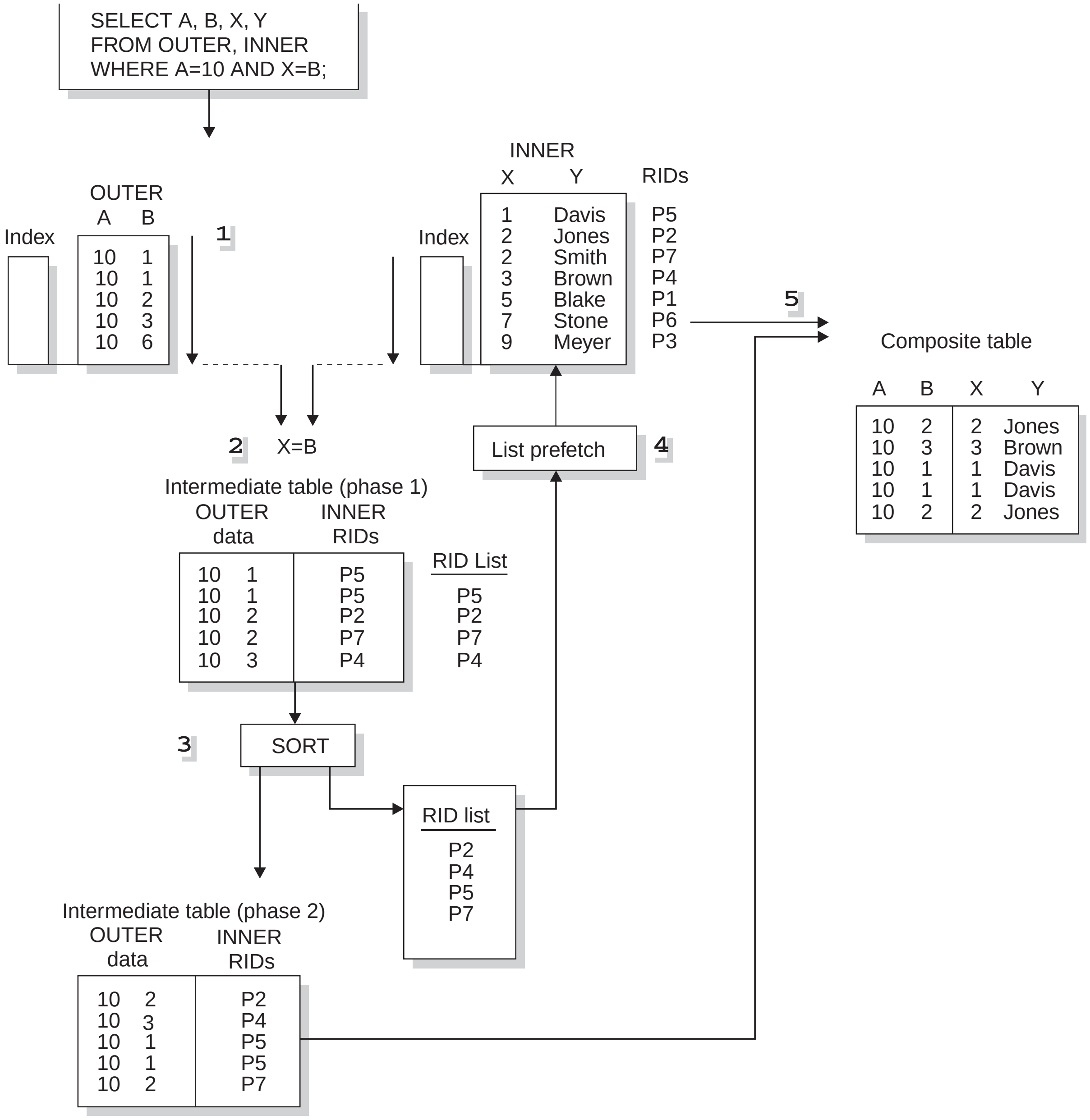


Figure 54. Hybrid join (SORTN_JOIN='Y')

The method requires obtaining RIDs in the order needed to use list prefetch. The steps are shown in Figure 54. In that example, both the outer table (OUTER) and the inner table (INNER) have indexes on the join columns.

DB2 performs the following steps:

1 Scans the outer table (OUTER).

2 Joins the outer table with RIDs from the index on the inner table. The result is the phase 1 intermediate table. The index of the inner table is scanned for every row of the outer table.

3 Sorts the data in the outer table and the RIDs, creating a sorted RID list and the phase 2 intermediate table. The sort is indicated by a value of Y in column

SORTN_JOIN of the plan table. If the index on the inner table is a well-clustered index, DB2 can skip this sort; the value in SORTN_JOIN is then N.

4 Retrieves the data from the inner table, using list prefetch.

5 Concatenates the data from the inner table and the phase 2 intermediate table to create the final composite table.

Possible EXPLAIN results for hybrid join

The following table shows possible EXPLAIN results from a hybrid join and an explanation of each column value.

Table 138. Explanation of EXPLAIN results for a hybrid join

Column value	Explanation
METHOD='4'	A hybrid join was used.
SORTC_JOIN='Y'	The composite table was sorted.
SORTN_JOIN='Y'	The intermediate table was sorted in the order of inner table RIDs. A non-clustered index accessed the inner table RIDs.
SORTN_JOIN='N'	The intermediate table RIDs were not sorted. A clustered index retrieved the inner table RIDs, and the RIDs were already well ordered.
PREFETCH='L'	Pages were read using list prefetch.

Performance considerations for hybrid join

Hybrid join uses list prefetch more efficiently than nested loop join, especially if indexes exist on the join predicate with low cluster ratios. It also processes duplicates more efficiently because the inner table is scanned only once for each set of duplicate values in the join column of the outer table.

If the index on the inner table is highly clustered, it is unnecessary to sort the intermediate table (SORTN_JOIN=N). The intermediate table is placed in a table in memory rather than in a work file.

When hybrid join is used

Hybrid join is often used under the following situations.
- A non-clustered index or indexes are used on the join columns of the inner table.
- The outer table has duplicate qualifying rows.

PSPI

Star schema access

DB2 can use special join methods, such as star join and pair-wise join, to efficiently join tables that form a star schema.

PSPI

A *star schema* is a logical database design that is included in decision support applications. A star schema is composed of a *fact table* and a number of *dimension tables* that are connected to it. A dimension table contains several values that are given an ID, which is used in the fact table instead of all the values.

You can think of the fact table, which is much larger than the dimension tables, as being in the center. The fact table is surrounded by dimension tables. The result resembles a star formation. The following figure illustrates the star formation that is created by a star schema.

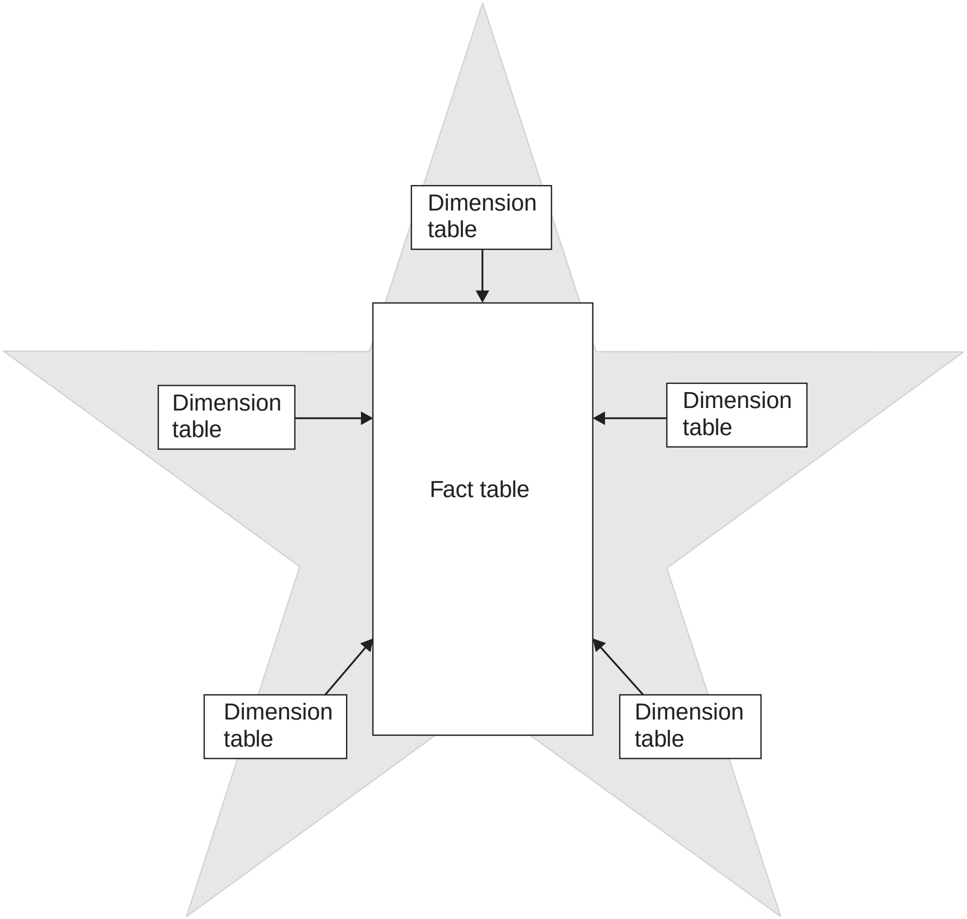


Figure 55. Star schema with a fact table and dimension tables

Unlike other join methods, such as nested loop join, merge scan join, and hybrid join, which join only two tables in each step, a single step in the star schema method can involve three or more tables. If the required indexes exist, DB2 might choose special methods such as star join and pair-wise join to process queries on a star schema more efficiently.

The index for the fact table must contain columns of only the following data types:

Example star schema

In this typical example, the star schema is composed of a fact table, SALES, and a number of dimension tables that are connected to it for time, products, and geographic locations.

The TIME table has a column for each month, quarter, and year. The PRODUCT table has columns for each product item, its class, and its inventory. The LOCATION table has columns of geographic data, including city and country.

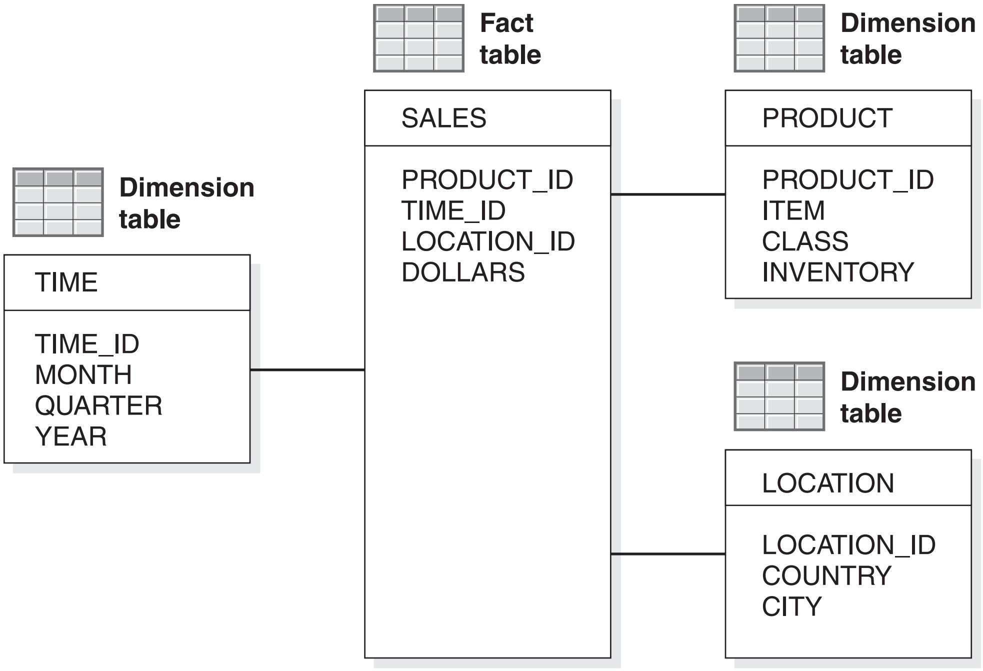


Figure 56. Example star schema with three dimension tables

In this scenario, the SALES table contains only three columns with IDs from the dimension tables, TIME, PRODUCT, and LOCATION, instead of three columns for time data, three columns for product data, and two columns for location data. Thus, the size of the fact table is greatly reduced. In addition, when you must change an item, you make only a single change in the dimension table, instead of making many changes in the fact table.

You can create even more complex star schemas by normalizing a dimension table into several tables. The normalized dimension table is called a *snowflake*. Only one of the tables in the snowflake joins directly with the fact table.

Example star join query with three dimension tables

PSPI

Suppose that you have a store in San Jose, and you want information about the sales of audio equipment from that store in 2005. For this example, you might join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You might write the following query to join the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, LOCATION L
  WHERE S.TIME = T.ID      AND
        S.PRODUCT = P.ID   AND
        S.LOCATION = L.ID  AND
        T.YEAR = 2005      AND
        P.CLASS = 'AUDIO'  AND
        L.LOCATION = 'SAN JOSE';
```

You would use the following index:

```
CREATE INDEX  XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

Your EXPLAIN output looks like the following table.

Table 139. Plan table output for a star join example with TIME, PRODUCT, and LOCATION

QUERYNO	QBLOCKNO	METHOD	TNAME	JOIN_ TYPE	SORTN_ JOIN	ACCESS TYPE
1	1	0	TIME	S	Y	R
1	1	1	PRODUCT	S	Y	R
1	1	1	LOCATION	S	Y	R
1	1	1	SALES	S		I

All snowflakes are processed before the central part of the star join, as individual query blocks, and are materialized into work files. A work file exists for each snowflake. The EXPLAIN output identifies these work files by naming them DSN_DIM_TBLX(*nn*), where *nn* indicates the corresponding QBLOCKNO for the snowflake.

This next example shows the plan for a star join that contains two snowflakes. Suppose that two new tables MANUFACTURER (M) and COUNTRY (C) are added to the tables in the previous example to break dimension tables PRODUCT (P) and LOCATION (L) into snowflakes:

- The PRODUCT table has a new column MID that represents the manufacturer.
- Table MANUFACTURER (M) has columns for MID and name to contain manufacturer information.
- The LOCATION table has a new column CID that represents the country.
- Table COUNTRY (C) has columns for CID and name to contain country information.

You might write the following query to join all the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, MANUFACTURER M,
       LOCATION L, COUNTRY C
  WHERE S.TIME = T.ID      AND
        S.PRODUCT = P.ID   AND
        P.MID = M.MID      AND
        S.LOCATION = L.ID  AND
        L.CID = C.CID      AND
        T.YEAR = 2005      AND
        M.NAME = 'some_company';
```

The joined table pairs (PRODUCT, MANUFACTURER) and (LOCATION, COUNTRY) are snowflakes. The EXPLAIN output of this query looks like the following table.

Table 140. Plan table output for a star join example with snowflakes

QUERYNO	QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE
1	1	0	TIME	S	Y	R
1	1	1	DSN_DIM_TBLX(02)	S	Y	R
1	1	1	SALES	S		I
1	1	1	DSN_DIM_TBLX(03)		Y	T
1	2	0	PRODUCT			R
1	2	1	MANUFACTURER			I
1	3	0	LOCATION			R
1	3	4	COUNTRY			I

Note: This query consists of three query blocks:

- QBLOCKNO=1: The main star join block
- QBLOCKNO=2: A snowflake (PRODUCT, MANUFACTURER) that is materialized into work file DSN_DIM_TBLX(02)
- QBLOCKNO=3: A snowflake (LOCATION, COUNTRY) that is materialized into work file DSN_DIM_TBLX(03)

The joins in the snowflakes are processed first, and each snowflake is materialized into a work file. Therefore, when the main star join block (QBLOCKNO=1) is processed, it contains four tables: SALES (the fact table), TIME (a base dimension table), and the two snowflake work files.

In this example, in the main star join block, the star join method is used for the first three tables (as indicated by S in the JOIN_TYPE column of the plan table) and the remaining work file is joined by the nested loop join with sparse index access on the work file (as indicated by T in the ACCESSTYPE column for DSN_DIM_TBLX(3)).

PSPI

When DB2 uses star schema access

To access the data in a star schema, you often write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables. DB2 uses star schema processing as the join type for the query if the following conditions are true:

- The number of tables in the star schema query block, including the fact table, dimensions tables, and snowflake tables, is greater than or equal to the value of the SJTABLES subsystem parameter.
- The value of subsystem parameter STARJOIN is 1, or the cardinality of the fact table to the largest dimension table meets the requirements that are specified by the value of the subsystem parameter. The values of STARJOIN and cardinality requirements are:

  DISABLED
  Star schema processing is disabled. This is the default.

ENABLED
Star schema processing is enabled if the cardinality of the fact table is at least 25 times the cardinality of the largest dimension that is a base table that is joined to the fact table.

1 Star schema processing is enabled. The one table with the largest cardinality is the fact table. However, if more than one table has this cardinality, star join is not enabled.

***n*, where 2 ≤ *n* ≤ 32768.**
Star schema processing is enabled if the cardinality of the fact table is at least *n* times the cardinality of the largest dimension that is a base table that is joined to the fact table.

In addition to the settings of SJTABLES and STARJOIN, the following conditions must also be met:

- The query references at least two dimensions.
- All join predicates are between the fact table and the dimension tables, or within tables of the same snowflake. If a snowflake is connected to the fact table, only one table in the snowflake (the central dimension table) can be joined to the fact table.
- All join predicates between the fact table and dimension tables are equijoin predicates.
- All join predicates between the fact table and dimension tables are Boolean term predicates.
- None of the predicates consist of a local predicate on a dimension table and a local predicate on a different table that are connected with an OR logical operator.
- No correlated subqueries cross dimensions.
- No single fact table column is joined to columns of different dimension tables in join predicates. For example, fact table column `F1` cannot be joined to column `D1` of dimension table T1 and also joined to column `D2` of dimension table T2.
- After DB2 simplifies join operations, no outer join operations exist.
- The data type and length of both sides of a join predicate are the same.
- The index on the fact table contains columns that have only the following data types:
  - CHARACTER
  - DATE
  - DECIMAL
  - DOUBLE
  - INTEGER
  - SMALLINT
  - REAL
  - TIME
  - TIMESTAMP
  - VARCHAR

Star join, which can reduce bind time significantly, does not provide optimal performance in all cases. The performance of star join depends on the availability of indexes on the fact table, the cluster ratio of the indexes, and the selectivity of local and join predicates, among other factors. Follow these general guidelines for setting the value of SJTABLES:

- If you have queries that reference fewer than 10 tables in a star schema database and you want to make the star join method applicable to all qualified queries, set the value of SJTABLES to the minimum number of tables that are used in queries that you want to be considered for star join.

  For example, suppose that you query a star schema database that has one fact table and three dimension tables. Set SJTABLES to 4.
- If you want to use star join for relatively large queries that reference a star schema database but are not necessarily suitable for star join, use the default. The star join method is considered for all qualified queries that have 10 or more tables.
- If you have queries that reference a star schema database but, in general, do not want to use star join, consider setting SJTABLES to a higher number, such as 15, if you want to drastically cut the bind time for large queries and avoid a potential bind time -101 SQL return code for large qualified queries.

Related concepts:

Boolean term predicates

How DB2 simplifies join operations

Indexes for efficient star schema processing

Related reference:

STAR JOIN QUERIES field (STARJOIN subsystem parameter) (DB2 Installation and Migration)

SJTABLES in macro DSN6SPRM (DB2 Installation and Migration)

Star join access (JOIN_TYPE='S'):

In *star join* processing, DB2 joins dimension tables to the fact table according to a multi-column index that is defined on the fact table.

Having a well-defined, multi-column index on the fact table is critical for efficient star join processing. Star join processing consolidates the dimensions to form a Cartesian product for matching to the single index on the fact table. Star join applies only to queries that qualify for star schema processing. This access method requires a single index on the fact table that supports the filtering that is provided by the dimension tables of the star schema.

If DB2 does not consider star join to be a cost effective access path for processing the star schema query, DB2 might choose pair-wise join (JOIN_TYPE='P'), or more-traditional join methods, such as nested loop, hybrid, or merge-scan.

For star join access to be considered, all columns in the index on the fact table must have one of the following data types:

- CHARACTER
- DATE
- DECIMAL
- DOUBLE
- INTEGER
- SMALLINT
- REAL
- TIME
- TIMESTAMP

- VARCHAR

Example: star schema with three dimension tables

PSPI

Suppose that you have a store in San Jose and want information about sales of audio equipment from that store in 2005. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, LOCATION L
  WHERE S.TIME = T.ID      AND
        S.PRODUCT = P.ID   AND
        S.LOCATION = L.ID  AND
        T.YEAR = 2005      AND
        P.CLASS = 'AUDIO'  AND
        L.LOCATION = 'SAN JOSE';
```

You would use the following statement to create an index:

```
CREATE INDEX XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

The following table shows the EXPLAIN output for this example.

Table 141. Plan table output for a star join example with TIME, PRODUCT, and LOCATION

QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE	PRIMARY ACCESS TYPE
1	0	TIME	S	Y	R	
1	1	PRODUCT	S	Y	R	T
1	1	LOCATION	S	Y	R	T
1	1	SALES	S		I	

In the example above, all dimensions were accessed before the fact table. However, DB2 might choose to access only certain filtering dimensions before accessing the fact table, and remaining dimensions might be joined later. This is a cost-based decision that is made by optimizer.

Example: star schema with two snowflakes

All snowflakes are processed before the central part of a star join, as individual query blocks, and they are materialized into work files. A work file exists for each snowflake. The EXPLAIN output identifies these work files by naming them DSN_DIM_TBLX(*nn*), where *nn* indicates the corresponding QBLOCKNO for the snowflake.

Suppose that two new tables MANUFACTURER (M) and COUNTRY (C) are added to the tables in the previous example to break dimension tables PRODUCT (P) and LOCATION (L) into snowflakes:

- The PRODUCT table has a new column MID that represents the manufacturer.
- The MANUFACTURER table has columns for MID and name to contain manufacturer information.
- The LOCATION table has a new column CID that represents the country.
- The COUNTRY table has columns for CID and name to contain country information.

You could write the following query to join all the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, MANUFACTURER M,
       LOCATION L, COUNTRY C
  WHERE S.TIME = T.ID       AND
        S.PRODUCT = P.ID    AND
        P.MID = M.MID       AND
        S.LOCATION = L.ID   AND
        L.CID = C.CID       AND
        T.YEAR = 2005       AND
        M.NAME = 'some_company';
```

The joined table pairs (PRODUCT, MANUFACTURER and LOCATION, COUNTRY) are snowflakes. The following table shows the EXPLAIN output for this example.

Table 142. Plan table output for a star join example with snowflakes

QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE	PRIMARY ACCESS TYPE
1	0	TIME	S	Y	R	
1	1	DSN_DIM_TBLX(02)	S	Y	R	T
1	1	SALES	S		I	
1	1	DSN_DIM_TBLX(03)		Y	R	T
2	0	PRODUCT			R	
2	1	MANUFACTURER			I	
3	0	LOCATION			R	
3	4	COUNTRY			I	

The sample explain output above shows that this query consists of several query blocks:

QBLOCKNO=1
The main star join block

QBLOCKNO=2
A snowflake (PRODUCT, MANUFACTURER) that is materialized into work file DSN_DIM_TBLX(02)

QBLOCKNO=3
A snowflake (LOCATION, COUNTRY) that is materialized into work file DSN_DIM_TBLX(03)

The joins in the snowflakes are processed first, and each snowflake is materialized into a work file. Therefore, when the main star join block (QBLOCKNO=1) is processed, it contains four tables: SALES (the fact table), TIME (a base dimension table), and the two snowflake work file tables.

In this example, in the main star join block, the star join method is used for the first three tables (as indicated by S in the JOIN TYPE column of the plan table). The remaining work file is joined by the nested loop join with sparse index access on the work file (as indicated by T in the PRIMARY_ACCESSTYPE column for DSN_DIM_TBLX(3)).

PSPI

Pair-wise join access (JOIN_TYPE='P'):

In *pair-wise* join processing DB2 matches each dimension to fact table independently, according to a single-column index for each dimension table.

RID lists from each join to the fact table index are intersected in pairs. The first pair of RID lists are intersected, the result of that intersection is paired with the next RID list and so on, until all viable dimension filtering is intersected. Pair-wise join applies only to queries that qualify for star schema processing. This access method requires separate indexes on the fact table that support each individual filtering dimension tables of the star schema. Pair-wise join requires a separate fact table index for each filtering dimension.

If DB2 does not consider pair-wise join to be a cost effective access path for processing the star schema query, DB2 might choose star join (JOIN_TYPE='S') or more-traditional join methods, such nested loop join, hybrid join, or merge-scan join.

For star join access to be considered, all columns in the index on the fact table must have one of the following data types:

- CHARACTER
- DATE
- DECIMAL
- DOUBLE
- INTEGER
- SMALLINT
- REAL
- TIME
- TIMESTAMP
- VARCHAR

Example:

PSPI

Suppose that you have a store in San Jose and want information about sales of audio equipment from that store in 2005. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year

- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You can write the following query to join the tables:

```
SELECT *
  FROM SALES S, TIME T, PRODUCT P, LOCATION L
 WHERE S.TIME = T.ID
   AND S.PRODUCT = P.ID
   AND S.LOCATION = L.ID
   AND T.YEAR = 2005
   AND P.CLASS = 'AUDIO'
   AND L.LOCATION = 'SAN JOSE';
```

Instead of creating a single index on the fact table that supports the various combinations of filtering, you can create separate indexes on the fact table for each dimension table. For this example you would use the following three statements to create the indexes:

```
CREATE INDEX XSALES_T ON SALES (TIME);
CREATE INDEX XSALES_P ON SALES (PRODUCT);
CREATE INDEX XSALES_L ON SALES (LOCATION);
```

The following table shows the EXPLAIN output for this example.

Table 143. Plan table output for a pair-wise join example with TIME, PRODUCT, and LOCATION

QBLOCKNO	METHOD	TNAME	JOIN TYPE	ACCESS NAME	ACCESS TYPE	MIXOPSEQ
1	0	SALES	P		P	0
1	0	SALES	P	DSNPWJ(06)	MX	1
1	0	SALES	P	DSNPWJ(08)	MX	2
1	0	SALES	P		MI	3
1	0	SALES	P	DSNPWJ(10)	MX	4
1	0	SALES	P		MI	5
1	0	DSN_DIM_TBLX(02)			R	6
1	1	SALES		XSALES_T	I	7
1	0	DSN_DIM_TBLX(03)			R	8
1	1	SALES		XSALES_P	I	9
1	0	DSN_DIM_TBLX(04)			R	10
1	1	SALES		XSALES_L	I	11
1	1	DSN_DIM_TBLX(02)			R	
1	1	DSN_DIM_TBLX(03)			R	
1	1	DSN_DIM_TBLX(04)			R	
2	0	TIME			R	
3	0	PRODUCT			R	
4	0	LOCATION			R	

Dimension tables chosen for pair-wise join before the fact table must be materialized into their own dimension tables similar to snowflakes in star join access method. Query block 2, 3 and 4 build the work files for TIME, PRODUCT and LOCATION dimensions respectively. These dimensions are accessed before the fact table, and are also required to be joined back after the fact table, to ensure data consistency.

The sample EXPLAIN output above shows the following multi-index access steps:

MIXOPSEQ = 0
Access the data pages for the fact table

MIXOPSEQ = 1
ACCESSNAME='DSNPWJ(06)' represents the pair-wise RID list built from multi-index access steps 6 and 7

MIXOPSEQ = 2
ACCESSNAME='DSNPWJ(08)' represents the pair-wise RID list built from multi-index access steps 8 and 9

MIXOPSEQ = 3
Intersection of RID lists from steps 1 and 2

MIXOPSEQ = 4
ACCESSNAME='DSNPWJ(10)' represents the pair-wise RID list built from multi-index access steps 10 and 11

MIXOPSEQ = 5
Intersection of RID lists from steps 3 and 4

MIXOPSEQ = 6
Materialized dimension table, DSN_DIM_TBLX(02), built from query block 2

MIXOPSEQ = 7
Join from step 6 to the fact table index XSALES_T to build RID list as input to step 1

MIXOPSEQ = 8
Materialized dimension table DSN_DIM_TBLX(03) built from query block 3

MIXOPSEQ = 9
Join from step 6 to the fact table index XSALES_P to build RID list as input to step 2

MIXOPSEQ = 10
Materialized dimension table DSN_DIM_TBLX(04) built from query block 4

MIXOPSEQ = 11
Join from step 6 to the fact table index XSALES_L to build RID list as input to step 4

Based upon cost, DB2 might choose to access one or more dimensions before the fact table because each dimension has a corresponding fact table index.

Enabling data caching for star schema queries:

You can enable data caching to improve the performance of queries on star schemas.

About this task

PSPI

When data caching is enabled for star schema queries, DB2 caches data from work files that are used by star schema queries. Data caching provides the following advantages:

Immediate data availability
During a star join operation, work files might be scanned many times. If

the work file data is cached in the dedicated virtual memory pool, that data is immediately available for join operations.

Reduced buffer pool contention
Because the virtual memory space for data caching is separated from the work file buffer pool, contention with the buffer pool is reduced. Reduced contention improves performance particularly when sort operations are performed concurrently.

The default virtual memory pool size is 20 MB. To set the pool size, use the SJMX, or DXPOOL parameter on the DSNTIP8 installation panel.

Procedure

To determine the best setting of the MAX DATA CACHING parameter for star schema queries:

1. Determine the value of A. Estimate the average number of work files that a star schema query uses. In typical cases, with highly normalized star schemas, the average number is about three to six work files.
2. Determine the value of B. Estimate the number of work file rows, the maximum length of the key, and the total of the maximum length of the relevant columns. Multiply these three values together to find the size of the data caching space for the work file, or the value of B.
3. Multiply (A) × (B) to determine the size of the pool in MB.

Example

The following example shows how to determine the size of the virtual memory for data caching. Suppose that you issue the following star join query, where SALES is the fact table:

```
SELECT C.COUNTRY, P.PRDNAME, SUM(F.SPRICE)
  FROM SALES F, TIME T, PROD P, LOC L, SCOUN C
  WHERE F.TID = T.TID AND
        F.PID = P.PID AND
        F.LID = L.LID AND
        L.CID = C.CID AND
        P.PCODE IN (4, 7, 21, 22, 53)
  GROUP BY .COUNTRY, P.PRDNAME;
```

The following table shows the EXPLAIN output for this example query.

Table 144. EXPLAIN output for a star schema query

QBLOCK NO	PLAN NO	TNAME	METHOD	JOIN_ TYPE	ACCESS TYPE	ACCESS NAME	PRIMARY ACCESS TYPE
1	1	TIME	0	S	R		
1	2	PROD	1	S	R		T
1	3	SALES	1	S	I	XSALES	
1	4	DSN_DIM_TBLX(02)	1		R		T
1	5		3				
2	1	LOC	0		R		
2	2	SCOUN	4		I	XSCOUN	

For this query, two work files can be cached in memory. These work files, PROD and DSN_DIM_TBLX(02), are indicated by PRIMARY_ACCESSTYPE=T.

1. In this example, the star join query uses two work files, PROD and DSN_DIM_TBLX(02). Therefore B = 2.
2. Both PROD and DSN_DIM_TBLX(02) are used to determine the value of B.

   Recommendation: Average the values for a representative sample of work files, and round the value up to determine an estimate for a value of C:

   - The number of work file rows depends on the number of rows that match the predicate. For PROD, 87 rows are stored in the work file because 87 rows match the IN-list predicate. No selective predicate is used for DSN_DIM_TBLX(02), so the entire result of the join is stored in the work file. The work file for DSN_DIM_TBLX(02) holds 2800 rows.
   - The maximum length of the key depends on the data type definition of the table's key column. For PID, the key column for PROD, the maximum length is 4. DSN_DIM_TBLX(02) is a work file that results from the join of LOC and SCOUN. The key column that is used in the join is LID from the LOC table. The maximum length of LID is 4.
   - The maximum data length depends on the maximum length of the key column and the maximum length of the column that is selected as part of the star join. Add to the maximum data length 1 byte for nullable columns, 2 bytes for varying length columns, and 3 bytes for nullable and varying length columns.

     For the PROD work file, the maximum data length is the maximum length of PID, which is 4, plus the maximum length of PRDNAME, which is 24. Therefore, the maximum data length for the PROD work file is 28. For the DSN_DIM_TBLX(02) work file, the maximum data length is the maximum length of LID, which is 4, plus the maximum length of COUNTRY, which is 36. Therefore, the maximum data length for the DSN_DIM_TBLX(02) work file is 40.

   Consequently, for PROD, B = (87) ×(4 + 28) = 2784 bytes. For DSN_DIM_TBLX(02), B = (2800) × (4 + 40) = 123200 bytes.

   The average of these two estimated values for B is approximately 62 KB. Because the number of rows in each work file can vary depending on the selection criteria in the predicate, the value of B should be rounded up to the nearest multiple of 100 KB. Therefore B = 100 KB.
3. The size of the pool is determined by multiplying (B) × (C) or, in this example, (2) × (100 KB) = 0.2 MB.

PSPI

Subquery access

The EXPLAIN output in the PLAN_TABLE sometimes shows the position and order in which subqueries are executed.

PSPI

The subqueries are indicated by a row in the PLAN_TABLE having TNAME="DSNWFQB(*nn*)", where '*nn*' is the query block number associated with the subquery, and TABLETYPE='S'. In PLAN_TABLE, the PARENT_PLANNO column corresponds to the plan number in the parent query block where the correlated subquery is invoked for correlated subqueries. For non-correlated subqueries it corresponds to the plan number in the parent query block that represents the work file for the subquery.

Non-correlated subqueries

The EXPLAIN output below is for the non-correlated form of the following subquery:

```
SELECT * FROM T1 WHERE T1.C2 IN (SELECT T2.C2 FROM T2, T3 WHERE T2.C1 = T3.C1)
```

Table 145. EXPLAIN output for the non-correlated subquery

QB NO	PLAN NO	METHOD	TNAME	AC TYPE	MC	AC NAME	SC_ JN	PAR_ QB	PAR_ PNO	QB TYPE	TB TYPE
1	1	0	DSNWFQB(02)	R	0		N	0	0	SELECT	W
1	2	1	T1	I	1	T1_1X_C2	Y	0	0	SELECT	T
2	1	0	T2	R	0		N	1	1	NCOSUB	T
2	2	1	T3	I	1	T3_X_C1	N	1	1	NCOSUB	T

In the example above, the row corresponding to QBNO=2 and PLANNO=1 has PARENT_PLANNO (abbreviated as PAR_PNO) = 1 and PARENT_QBNO (abbreviated as PAR_QB) = 1. This means that the row corresponding to QBNO=1 and PLANNO=1 is the parent row. The sequence of execution flows from parent to child, then back to parent after the child rows are exhausted. In the example above that means the sequence of execution is (QBNO, PLANNO): (1,1) , (2,1), (2,2), (1,2).

Correlated subqueries

The following example shows a correlated subquery and the associated EXPLAIN output:

```
SELECT * FROM T1
  WHERE EXISTS (SELECT 1 FROM T2, T3
                WHERE T2.C1 = T3.C1 AND T2.C2 = T1.C2)
```

Table 146. EXPLAIN output for the correlated subquery:

QB NO	PLAN NO	METHOD	TNAME	AC TYPE	MC	ACNAME	SC_ JN	PAR_ QB	PAR_ PNO	QBTYPE	TB TYPE
1	10	0	T1	R	0		N	0	0	SELECT	T
2	1	1	T2	I	1	T2_IX_C2	N	1	1	CORSUB	T
2	2	1	T3	I	1	T3_IX_C1	N	1	1	CORSUB	T

Subqueries transformed to joins

A plan table shows that a subquery is transformed into a join by the value in column QBLOCKNO.

- If the subquery is not transformed into a join, that means it is executed in a separate operation, and its value of QBLOCKNO is greater than the value for the outer query.
- If the subquery is transformed into a join, it and the outer query have the same value of QBLOCKNO. A join is also indicated by a value of 1, 2, or 4 in column METHOD.

PSPI

Related concepts:
Investigating join operations

Subqueries (DB2 Application programming and SQL)

Related tasks:
Writing efficient subqueries

View and nested table expression access

A nested *table expression* is the specification of a subquery in the FROM clause of an SQL SELECT statement. The processing of table expressions is similar that for a view.

PSPI

You can determine the methods that are used by executing EXPLAIN for the statement that contains the view or nested table expression. In addition, you can use EXPLAIN to determine when set operators are used and how DB2 might eliminate unnecessary subselect operations to improve the performance of a query.

PSPI

Materialization for views and nested table expressions

Materialization means that the data rows that are selected by the view or nested table expression are put into a work file that is to be processed like a table. When the column TNAME names a view or nested table expression and column TABLE_TYPE contains a W, it indicates that the view or nested table expression is materialized.

Merge processing

In *merge* processing, a statement that references a view or table expression is combined with the fullselect that defined the view or table expression. This combination creates a logically equivalent statement. This equivalent statement is executed against the database.

PSPI

The merge process is more efficient than materialization.

Important: The merge process and the MERGE statement are not related concepts, and must not be confused.

Example: opportunity for merge processing

Consider the following statements, one of which defines a view, the other of which references the view:

```
View-defining statement:                View referencing statement:

CREATE VIEW VIEW1 (VC1,VC21,VC32) AS    SELECT VC1,VC21
  SELECT C1,C2,C3 FROM T1                 FROM VIEW1
    WHERE C1 > C3;                        WHERE VC1 IN (A,B,C);
```

The fullselect of the view-defining statement can be merged with the view-referencing statement to yield the following logically equivalent statement:

```
Merged statement:

SELECT C1,C2 FROM T1
  WHERE C1 > C3 AND C1 IN (A,B,C);
```

Merge processing for correlated table expressions

DB2 can perform merge processing for correlated table expressions.

```
SELECT *
 FROM T1,
  TABLE(
   SELECT T1.C2 from T3 AS T2
   WHERE T1.C1 = T2.C1
   ) AS X;
```

DB2 does not materialize the table expression in this query. The expression is merged into the parent query block, and the query is rewritten as the following query:

```
SELECT *
FROM T1, T3 AS T2
WHERE T1.C1 = T2.C1
```

Merge processing for views and table expressions with subqueries on outer joins

DB2 can avoid materialization for queries that have table expressions and views either side of left and right outer joins. The kind of merge processing that DB2 can use depends on which side of the outer join contains the view or table expression, and whether the query contains a subquery.

For example, the following query uses a left outer join and contains reference to a view on the left side of the join:

```
CREATE VIEW V1
 AS SELECT C1, C2
   FROM T3
   WHERE T3.C1 IN (SELECT T4.C1
          FROM T4 WHERE T4.C2 = T3.C2
          GROUP BY T4.C1);
SELECT T2.C1, T2.C2, T1.C1,T2.C2
FROM V1 AS T1
 LEFT OUTER JOIN
 T2 ON T1.C1= T2.C1
 WHERE (T1.C2 IN('712' , '713', '714'));
```

In such cases, where the view or table expression is on the preserved row side of the of the join, DB2 does not materialize the view so that selective predicates such as `T1.C2 IN('712' , '713', '714')` can be applied earlier and reduce the size of the join.

Similarly, if the view or table expression is on the null-supplying side of a left or right outer join, and the following conditions are also true, DB2 merges the table expression:

- The view or table expression contains a subquery.
- The view or table expression contains references to only a single table.

For example, DB2 merges the table expression for the following query:

```
SELECT *
FROM T1
 LEFT OUTER JOIN
  (SELECT *
   FROM T2
   WHERE T2.C1 = (SELECT MAX(T3.C1) FROM T3 )
  ) TE
 ON T1.C1 = TE.C1;
```

DB2 merges the table expression by converting the subquery predicate into a before-join predicate to avoid materialization. For example:

```
SELECT *
FROM T1
 LEFT OUTER JOIN
 T2 as TT
 ON TT.C1 = (SELECT MAX(TTT.C1)
       FROM T3 AS TTT)
       AND T1.C1 = TT.C1;
```

Merge processing in statements with CASE, VALUE, NULLIF, IFNULL, and COALESCE expressions

If there are CASE, VALUE, NULLIF, IFNULL, or COALESCE expressions on the preserved side of an outer join, DB2 can merge the view or table instead of materializing the view or table.

```
SELECT A.C1, B.C1, A.C2, B.C2
FROM T1 ,(SELECT COALESCE(C1, 0) ,C2
 FROM T2 ) A(C1,C2)
  LEFT OUTER JOIN
   (SELECT COALESCE(C1, 0) ,C2
   FROM T3 ) B(C1,C2)
  ON A.C2 = B.C2
WHERE T1.C2 = A.C2;
```

In this case, DB2 merges table expression A, but materializes table expression B to apply the COALESCE operation before supplying nulls.

More examples

The following statements show another example of when a view and table expression can be merged:

```
SELECT * FROM V1 X
  LEFT JOIN
    (SELECT * FROM T2) Y ON X.C1=Y.C1
       LEFT JOIN T3 Z ON X.C1=Z.C1;
```

```
Merged statement:

SELECT * FROM V1 X
  LEFT JOIN
    T2 ON X.C1 = T2.C1
      LEFT JOIN T3 Z ON X.C1 = Z.C1;
```

PSPI

Materialization

Views and table expressions cannot always be merged. In certain cases, DB2 materializes the view or table expression

PSPI

Introductory concepts

DB2 views (Introduction to DB2 for z/OS)

In the following example, DB2 performs materialization of the view or table expression, which is a two step process.

1. The fullselect that defines the view or table expression is executed against the database, and the results are placed in a temporary copy of a result table.
2. The statement that references the view or table expression is then executed against the temporary copy of the result table to obtain the intended result.

Whether materialization is needed depends upon the attributes of the referencing statement, or logically equivalent referencing statement from a prior merge, and the attributes of the fullselect that defines the view or table expression.

Example

Look at the following statements:

View defining statement

```
CREATE VIEW VIEW1 (VC1,VC2) AS
  SELECT SUM(C1),C2 FROM T1
    GROUP BY C2;
```

View referencing statement

```
SELECT MAX(VC1)
  FROM VIEW1;
```

Column VC1 occurs as the argument of an aggregate function in the view referencing statement. The values of VC1, as defined by the view-defining fullselect, are the result of applying the aggregate function SUM(C1) to groups after grouping the base table T1 by column C2. No equivalent single SQL SELECT statement can be executed against the base table T1 to achieve the intended result. You cannot specify that aggregate functions be applied successively.

PSPI

When views and nested table expressions are materialized

DB2 uses materialization to satisfy a reference to a view or table expression when aggregate processing is involved (such grouping, aggregate functions, and distinct operations).This processing is indicated by the defining fullselect, with either aggregate processing indicated by the statement references the view or table expression, or by the view or table expression that participates in a join. For views and table expressions that are defined with set operators, DB2 can often distribute aggregate processing, joins, and qualified predicates to avoid materialization.

The following table indicates some cases in which materialization occurs. DB2 can also use materialization in statements that contain multiple outer joins, outer joins that combine with inner joins, or merges that cause a join of greater than 15 tables.

Table 147. Cases when DB2 performs view or table expression materialization. Each “X” indicates a case of materialization.

SELECT FROM view or table expression uses...[1]	View definition or table expression[2] uses GROUP BY	View definition or table expression[2] uses DISTINCT	View definition or table expression[2] uses Aggregate function	View definition or table expression[2] uses Aggregate function DISTINCT	View definition or table expression[2] uses UNION	View definition or table expression[2] uses UNION ALL[4]
Joins [3]	X	X	X	X	X	
GROUP BY	X	X	X	X	X	
DISTINCT		X		X	X	
Aggregate function	X	X	X	X	X	X
Aggregate function DISTINCT	X	X	X	X	X	
SELECT subset of view or table expression columns		X			X	

Notes:

1. If the view is referenced as the target of an insert, update, or delete operation to satisfy the view reference. Only updatable views can be the target in insert, update, and delete operations.

   An SQL statement can reference a particular view multiple times where some of the references can be merged and some must be materialized.

2. If a SELECT list contains a host variable in a table expression, then materialization occurs. For example:

   ```
   SELECT C1 FROM
      (SELECT :HV1 AS C1 FROM T1) X;
   ```

   If a view or nested table expression is defined to contain a user-defined function, and if that user-defined function is defined as NOT DETERMINISTIC or EXTERNAL ACTION, then the view or nested table expression is always materialized.

3. Additional details about materialization with outer joins:

   - If a WHERE clause exists in a view or table expression, and it does not contain a column, materialization occurs.

     ```
     SELECT X.C1 FROM
        (SELECT C1 FROM T1
          WHERE 1=1) X LEFT JOIN T2 Y
                     ON X.C1=Y.C1;
     ```

   - If the outer join is a full outer join and the SELECT list of the view or nested table expression does not contain a standalone column for the column that is used in the outer join ON clause, then materialization occurs.

     ```
     SELECT X.C1 FROM
        (SELECT C1+10 AS C2 FROM T1) X FULL JOIN T2 Y
                     ON X.C2=Y.C2;
     ```

   - If the SELECT list of a view or nested table expression contains no column, materialization occurs.

```
SELECT X.C1 FROM
   (SELECT 1+2+:HV1. AS C1 FROM T1) X LEFT JOIN T2 Y
                  ON X.C1=Y.C1;
```

   - If certain conditions are met, materialization can be avoided when a left or right outer join contains the following types of expressions: CASE, COALESCE, or VALUE.

     If the preserved-row view or table expression (the left side of a left join) contains a CASE, COALESCE, or VALUE expression, the view or table expression is materialized only if the expression is referenced in an ON or WHERE clause predicate outside of the table expression. A reference in the select list to the CASE, COALESCE, or VALUE expression on the preserved-row side does not cause materialization.

     However, if the null-supplied view or table expression (the right side in a left join) contains a CASE, COALESCE, or VALUE expression, the view or table expression is materialized if the expression is referenced as a predicate, or in the select list of the outer query.

     For example, consider the following statement:

     ```
     SELECT A.C1, B.C1, A.C2, B.C2
     FROM T1 ,(SELECT COALESCE(C1, 0) AS C1 ,C2 FROM T2 ) A
      LEFT OUTER JOIN
      (SELECT COALESCE(C1, 0) AS C1 ,C2 FROM T3 ) B
      ON A.C2 = B.C2
     WHERE T1.C2 = A.C2;
     ```

     A is the preserved table expression of a left outer join. Because A.C1 is not referenced by any predicate, materialization can be avoided for table expression A.

     B is the null-supplied table expression of the left outer join. Because B.C1 is referenced in the select list for the statement, table expression B must be materialized. If any predicate contained a reference to B.C1, that would also require materialization of table expression B.

4. DB2 cannot avoid materialization for UNION ALL in all cases. Some of the situations in which materialization occurs includes:
   - When the view is the operand in an outer join for which nulls are used for non-matching values, materialization occurs. This situation happens when the view is either operand in a full outer join, the right operand in a left outer join, or the left operand in a right outer join.
   - If the number of tables would exceed 225 after distribution, then distribution does not occur, and the result is materialized.
5. For INTERSECT and EXCEPT set operators, the EXPLAIN information might help to determine if the view is materialized.

PSPI

Related concepts:

Left outer join (DB2 Application programming and SQL)

Right outer join (DB2 Application programming and SQL)

Performance of merge versus materialization

The merge process, which is not related to MERGE statements in SQL, performs better than materialization.

PSPI

For materialization, DB2 uses a table space scan to access the materialized temporary result. DB2 also uses default statistics for the columns of the materialized temporary result, which might impact the selected access path.DB2 materializes a view or table expression only if it cannot use merge processing.

Materialization is a two-step process with the first step resulting in the formation of a temporary result. The smaller the temporary result, the more efficient is the second step. To reduce the size of the temporary result, DB2 attempts to evaluate certain predicates from the WHERE clause of the referencing statement at the first step of the process rather than at the second step. Only certain types of predicates qualify. First, the predicate must be a simple Boolean term predicate. Second, it must have one of the forms shown in Table 148.

Table 148. Predicate candidates for first-step evaluation

Predicate	Example
COL op constant	V1.C1 > hv1
COL IS (NOT) NULL	V1.C1 IS NOT NULL
COL (NOT) BETWEEN constant AND constant	V1.C1 BETWEEN 1 AND 10
COL (NOT) LIKE constant (ESCAPE constant)	V1.C2 LIKE 'p\%%' ESCAPE '\'
COL IN (list)	VI.C2 IN (a,b,c)

Note: Where "op" is =, <>, >, <, <=, or >=, and constant is either a host variable, constant, or special register. The constants in the BETWEEN predicate need not be identical.
Implied predicates generated through predicate transitive closure are also considered for first step evaluation.

PSPI

Using EXPLAIN to determine when materialization occurs

Rows that describe the access path for both steps of the materialization process, for each reference to a view or table expression that is materialized, appear in the PLAN_TABLE.

PSPI

These rows describe the access path used to formulate the temporary result indicated by the defining fullselect of the view, and they describe the access to the temporary result as indicated by the referencing statement. The defining fullselect can also refer to views or table expressions that need to be materialized.

When DB2 chooses materialization, TNAME contains the name of the view or table expression, and TABLE_TYPE contains a W. A value of Q in TABLE_TYPE for the name of a view or nested table expression indicates that the materialization was virtual and not actual. (Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.) When DB2 chooses merge, EXPLAIN data for the merged statement appears in PLAN_TABLE; only the names of the base tables on which the view or table expression is defined appear.

Example

Consider the following statements, which define a view and reference the view:

```
View defining statement:

CREATE VIEW V1DIS (SALARY, WORKDEPT) as
  (SELECT DISTINCT SALARY, WORKDEPT FROM DSN8810.EMP)

View referencing statement:

SELECT * FROM DSN8810.DEPT
  WHERE DEPTNO IN (SELECT WORKDEPT FROM V1DIS)
```

The following table shows a subset of columns in a plan table for the query.

Table 149. Plan table output for an example with view materialization

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
1	1	SELECT	DEPT	T	0
2	1	NOCOSUB	V1DIS	W	0
2	2	NOCOSUB		?	3
3	1	NOCOSUB	EMP	T	0
3	2	NOCOSUB		?	3

Notice how TNAME contains the name of the view and TABLE_TYPE contains W to indicate that DB2 chooses materialization for the reference to the view because of the use of SELECT DISTINCT in the view definition.

Example

Consider the following statements, which define a view and reference the view:

```
View defining statement:

CREATE VIEW V1NODIS (SALARY, WORKDEPT) as
  (SELECT SALARY, WORKDEPT FROM DSN8810.EMP)

View referencing statement:

SELECT * FROM DSN8810.DEPT
   WHERE DEPTNO IN (SELECT WORKDEPT FROM V1NODIS)
```

If the VIEW is defined without DISTINCT, DB2 chooses merge instead of materialization. In the sample output, the name of the view does not appear in the plan table, but the table name on which the view is based does appear.

The following table shows a sample plan table for the query.

Table 150. Plan table output for an example with view merge

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
1	1	SELECT	DEPT	T	0
2	1	NOCOSUB	EMP	T	0
2	2	NOCOSUB		?	3

When DB2 avoids materialization in such cases, TABLE_TYPE contains a Q to indicate that DB2 uses an intermediate result that is not materialized, and TNAME shows the name of this intermediate result as DSNWFQB(*xx*), where *xx* is the number of the query block that produced the result.

PSPI

Using EXPLAIN to determine UNION, INTERSECT, and EXCEPT activity and query rewrite

For each reference to a view or table expression that is defined with a UNION ALL operator, DB2 might rewrite the query into a logically equivalent statement with improved performance.

PSPI

DB2 rewrites the queries in the following manner:

- Distributing qualified predicates, joins, and aggregations across the subselects of UNION ALL. Such distribution helps to avoid materialization. No distribution is performed for UNION, INTERSECT, EXCEPT, INTERSECT ALL, or EXCEPT ALL.
- Eliminating unnecessary subselects of a view or table expression that was created by a UNION ALL operation. For DB2 to eliminate subselects, the referencing query and the view or table definition must have predicates that are based on common columns.

The QBLOCK_TYPE column in the plan table indicates set operator activity. If a set operation was used, the column contains one of the values shown in the following table.

Table 151. Meanings of the set operator values for the QBLOCK_TYPE column of PLAN_TABLE

QBLOCK_TYPE value	Set operator
UNION	UNION
UNIONA	UNION ALL
INTERS	INTERSECT
INTERA	INTERSECT ALL
EXCEPT	EXCEPT
EXCEPTA	EXCEPT ALL

When the value of QBLOCK_TYPE is set to UNION, INTERSECT, or EXCEPT, the METHOD column on the same row is set to 3 and the SORTC_UNIQ column is set to 'Y' to indicate that a sort is necessary to remove duplicates. As with other views and table expressions, the plan table also shows when DB2 uses materialization instead of merge.

Example: Consider the following statements, which define a viewby using the UNION ALL operator, reference that view, and demonstrate how DB2 can rewrite the referencing statement.

The statement that defines the view uses data from three tables of weekly data to create the view:

```
CREATE VIEW V1 (CUSTNO, CHARGES, DATE) as
  SELECT CUSTNO, CHARGES, DATE
  FROM WEEK1
  WHERE DATE BETWEEN '01/01/2006' And '01/07/2006'
UNION ALL
```

```
   SELECT CUSTNO, CHARGES, DATE
    FROM WEEK2
    WHERE DATE BETWEEN '01/08/2006' And '01/14/2006'
  UNION ALL
    SELECT CUSTNO, CHARGES, DATE
    FROM WEEK3
    WHERE DATE BETWEEN '01/15/2006' And '01/21/2006';
```

Another statement references the view to find the average charges for each customer in California during the first and third Friday of January 2006:

```
SELECT V1.CUSTNO, AVG(V1.CHARGES)
  FROM CUST, V1
  WHERE CUST.CUSTNO=V1.CUSTNO
    AND CUST.STATE='CA'
    AND DATE IN ('01/07/2006','01/21/2006')
  GROUP BY V1.CUSTNO;
```

DB2 can rewrite the statement (assuming that CHARGES is defined as NOT NULL):

```
SELECT CUSTNO_U, SUM(SUM_U)/SUM(CNT_U)
  FROM
  ( SELECT WEEK1.CUSTNO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST, WEEK1
      Where CUST.CUSTNO=WEEK1.CUSTNO AND CUST.STATE='CA'
            AND DATE BETWEEN '01/01/2006' And '01/07/2006'
            AND DATE IN ('01/07/2006','01/21/2006')
      GROUP BY WEEK1.CUSTNO
    UNION ALL
    SELECT WEEK3.CUSTNO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST,WEEK3
      WHERE CUST.CUSTNO=WEEK3 AND CUST.STATE='CA'
            AND DATE BETWEEN '01/15/2006' And '01/21/2006'
            AND DATE IN ('01/07/2006','01/21/2006')
      GROUP BY WEEK3.CUSTNO
  ) AS X(CUSTNO_U,SUM_U,CNT_U)
GROUP BY CUSTNO_U;
```

The following table shows a subset of columns in a plan table for the query.

Table 152. Plan table output for an example with a view with UNION ALL operations

QBLOCKNO	PLANNO	TNAME	TABLE_ TYPE	METHOD	QBLOCK_ TYPE	PARENT_ QBLOCK
1	1	DSNWFQB(02)	Q	0		0
1	2		?	3		0
2	1		?	0	UNIONA	1
3	1	CUST	T	0		2
3	2	WEEK1	T	1		2
4	1	CUST	T	0		2
4	2	WEEK3	T	2		2

Notice how DB2 eliminates the second subselect of the view definition from the rewritten query and how the plan table indicates this removal by showing a UNION ALL for only the first and third subselect in the view definition. The Q in the TABLE_TYPE column indicates that DB2 does not materialize the view.

PSPI

Related concepts:

Predicate manipulation

Query transformations

Related reference:

subselect (DB2 SQL)

fullselect (DB2 SQL)

PLAN_TABLE

EXPLAIN (DB2 SQL)

Interpreting query parallelism

You can examine plan table data to determine whether DB2 chooses access paths that take advantage of parallel processing.

About this task

PSPI

To understand the likelihood that DB2 chooses parallelism, examine your PLAN_TABLE output. This information describes a method for examining PLAN_TABLE columns for parallelism and provides several examples.

PSPI

Procedure

To interpret EXPLAIN output for parallelism:

1. Determine the likelihood that DB2 chooses parallelism: For each query block (QBLOCKNO) in a query (QUERYNO), a non-null value in ACCESS_DEGREE or JOIN_DEGREE indicates that some degree of parallelism is planned.
2. Identify the parallel groups in the query:

   All steps (PLANNO) with the same value for ACCESS_PGROUP_ID, JOIN_PGROUP_ID, SORTN_PGROUP_ID, or SORTC_PGROUP_ID indicate that a set of operations are in the same parallel group. Usually, the set of operations involves various types of join methods and sort operations. Parallel group IDs can appear in the same row of PLAN_TABLE output, or in different rows, depending on the operation being performed.
3. Identify the parallelism mode. The PARALLELISM_MODE column indicates that type of parallelism that is planned. Within a query block, you cannot have a mixture of 'I' and 'C' parallel modes. However, a statement that uses more than one query block, such as a UNION, can have 'I' for one query block and 'C' for another. You can have a mixture of 'C' and 'X' modes in a query block, but not in the same parallel group.

   If the statement was bound while this DB2 is a member of a data sharing group, the PARALLELISM_MODE column can contain “X” even if only this one DB2 member is active. This lets DB2 take advantage of extra processing power that might be available at execution time. If other members are not available at execution time, then DB2 runs the query within the single DB2 member.

Examples

The following examples illustrate some of the PLAN_TABLE values that represent parallelism. In each case, all examples might have the PARALLELISM_MODE value.

Single table access

Assume that DB2 decides at bind time to initiate three concurrent requests to retrieve data from table T1. Part of PLAN_TABLE appears as shown in the following table. If DB2 decides not to use parallel operations for a step, ACCESS_DEGREE and ACCESS_PGROUP_ID contain null values.

Table 153. Part of PLAN_TABLE for single table access

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)

Nested loop join

Consider a query that results in a series of nested loop joins for three tables, T1, T2 and T3. T1 is the outermost table, and T3 is the innermost table. DB2 decides at bind time to initiate three concurrent requests to retrieve data from each of the three tables. Each request accesses part of T1 and all of T2 and T3. For the nested loop join method with sort, all the retrievals are in the same parallel group except for star join with ACCESSTYPE=T (sparse index). Part of PLAN_TABLE appears as shown in the following table:

Table 154. Part of PLAN_TABLE for a nested loop join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	1	3	1	3	1	(null)	(null)
T3	1	3	1	3	1	(null)	(null)

Merge scan join

Consider a query that causes a merge scan join between two tables, T1 and T2. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. The scan and sort of T1 occurs in one parallel group. The scan and sort of T2 occurs in another parallel group. Furthermore, the merging phase can potentially be done in parallel. Here, a third parallel group is used to initiate three concurrent requests on each intermediate sorted table. Part of PLAN_TABLE appears as shown in the following table:

Table 155. Part of PLAN_TABLE for a merge scan join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	d	(null)	(null)	d	(null)
T2	2	6	2	3	3	d	d

In a multi-table join, DB2 might also execute the sort for a composite that involves more than one table in a parallel task. DB2 uses a cost basis model to determine whether to use parallel sort in all cases. When DB2 decides to use parallel sort, SORTC_PGROUP_ID and SORTN_PGROUP_ID indicate the parallel group identifier. Consider a query that joins three tables, T1, T2, and T3, and uses a merge scan join between T1 and T2, and then between the composite and T3. If DB2 decides, based on the cost model, that all sorts in this query are to be performed in parallel, part of PLAN_TABLE appears as shown in the following table:

Table 156. Part of PLAN_TABLE for a multi-table, merge scan join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	2	6	2	6	3	1	2
T3	2	6	4	6	5	3	4

Hybrid join

Consider a query that results in a hybrid join between two tables, T1 and T2. Furthermore, T1 needs to be sorted; as a result, in PLAN_TABLE the T2 row has SORTC_JOIN=Y. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. Parallel operations are used for a join through a clustered index of T2.

Because the T2 RID can be retrieved by initiating concurrent requests on the clustered index, the joining phase is a parallel step. The retrieval of the T2 RID and the T2 rows are in the same parallel group. Part of PLAN_TABLE appears as shown in the following table:

Table 157. Part of PLAN_TABLE for a hybrid join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	4	6	2	6	2	1	(null)

Related concepts:

Parallel processing access (PARALLELISM_MODE='I', 'C', or 'X')

Related tasks:

Programming for parallel processing

Tuning parallel processing

Enabling parallel processing

Disabling query parallelism

Generating visual representations of access plans (IBM Data Studio)

Related reference:

PLAN_TABLE

Estimating the cost of SQL statements

You can use EXPLAIN to populate a statement table, *owner*. DSN_STATEMNT_TABLE, at the same time as your PLAN_TABLE is being populated.

About this task

PSPI

DB2 provides cost estimates, in service units and in milliseconds, for SELECT, INSERT, UPDATE, and DELETE statements, both static and dynamic. The estimates do not take into account several factors, including cost adjustments that are caused by parallel processing, or the use of triggers or user-defined functions.

Procedure

Use the information provided in the statement table to:

- Determine if a statement is not going to perform within range of your service-level agreements and to tune accordingly.

  DB2 puts its cost estimate into one of two *cost categories*: category A or category B. Estimates that go into cost category A are the ones for which DB2 has adequate information to make an estimate. That estimate is not likely to be 100% accurate, but is likely to be more accurate than any estimate that is in cost category B.

  DB2 puts estimates into cost category B when it is forced to use default values for its estimates, such as when no statistics are available, or because host variables are used in a query.
- Give a system programmer a basis for entering service-unit values by which to govern dynamic statements with predictive governing.

  PSPI

Cost categories

DB2 uses cost categories to differentiate estimates for which adequate information is available from those for which it is not.

PSPI

You probably wouldn't want to spend a lot of time tuning a query based on estimates that are returned in cost category B, because the actual cost could be radically different based on such things as what value is in a host variable, or how many levels of nested triggers and user-defined functions exist.

Similarly, if system administrators use these estimates as input into the resource limit specification table for governing (either predictive or reactive), they probably would want to give much greater latitude for statements in cost category B than for those in cost category A.

Because of the uncertainty involved, category B statements are also good candidates for reactive governing.

Cost category A

DB2 puts everything that doesn't fall into category B into category A.

Cost category B

DB2 puts a statement's estimate into cost category B when any of the following conditions exist:

- The statement has UDFs.
- Triggers are defined for the target table:
  - The statement uses an insert operation, and insert triggers are defined on the target table.
  - The statement uses an update operation, and update triggers are defined on the target table.
  - The statement uses a delete operation, and delete triggers are defined on the target table.
- The target table of a delete statement has referential constraints defined on it as the parent table, and the delete rules are either CASCADE or SET NULL.
- The WHERE clause predicate has one of the following forms:
  - COL op constant, and the constant is a host variable, parameter marker, or special register. The operator can be >, >=, <, <=, LIKE, or NOT LIKE.
  - COL BETWEEN constant AND constant where either constant is a host variable, parameter marker, or special register.
  - LIKE with an escape clause that contains a host variable.
- The cardinality statistics are missing for one or more tables that are used in the statement.
- A subselect in the SQL statement contains a HAVING clause.

PSPI

Retrieving rows from a statement table

You can use rows in the statement table to determine the cost of executing an SQL statement.

Procedure

PSPI

To retrieve rows in a statement table:

- Issue a SELECT statement to retrieve the columns of a statement table. The following example statement, which retrieves all rows about the statement that is represented by query number 13:

  ```
  SELECT * FROM JOE.DSN_STATEMNT_TABLE
     WHERE QUERYNO = 13;
  ```

- Join the rows of the statement table to the corresponding rows in the plan table. Certain columns in a statement table contain the same values as corresponding columns of the plan table for a given plan, including:
  - QUERYNO
  - APPLNAME
  - PROGNAME
  - COLLID
  - EXPLAIN_TIME
  - VERSION
  - SECTNOI

  The following example statement retrieves all columns from *user-ID*.PLAN_TABLE, and cost-related columns from *user-ID*.DSN_STATEMENTTABLE, for all rows that are related to the 'APPL1' application.

```
SELECT A.*, PROCMS, COST_CATEGORY
 FROM JOE.PLAN_TABLE A, JOE.DSN_STATEMNT_TABLE B
   WHERE A.APPLNAME = 'APPL1' AND
   A.APPLNAME = B.APPLNAME AND
   A.QUERYNO = B.QUERYNO AND
   A.SECTNOI = B.SECTNOI AND
   A.PROGNAME = B.PROGNAME AND
   A.COLLID   = B.COLLID AND
   A.EXPLAIN_TIME = B.EXPLAIN_TIME
   A.VERSION = B.VERSION
ORDER BY A.QUERYNO[1], A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

  1. If the a static SQL package was bound with the EXPLAIN(YES) option and contains more than one statement with the same value for QUERYNO, use the SECTNOI column in place of QUERYNO.

PSPI

Chapter 45. Analyzing concurrency

You can analyze concurrency for an application that experiences problems with lock contention.

About this task

PSPI

In Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, a report titled “Trace” corresponds to a single DB2 trace record; a report titled “Report” summarizes information from several trace records.

The example uses the following reports for problem analysis:
- Accounting report - long
- Locking suspension report
- Lockout report
- Lockout trace

In some cases, the initial and detailed stages of tracing and analysis presented in this information can be consolidated into one. In other cases, the detailed analysis might not be required at all.

Procedure

To investigate concurrency and locking and latching problems:

1. Determine a period when the transaction is likely to encounter performance problems.
2. When the period begins, start the GTF.
3. Start the following traces, and specify the GTF as the destination:
   - Accounting Class(1, 2, 3, 7, 8).
   - Statistics Class(*)
   - Performance Class(1,2,3,6,7).

   You can specify the plan name, authorization ID, and package name associated with the problem transaction to reduce the volume of the trace information.
4. Start the DB2 accounting classes 1, 2, and 3 to GTF to allow for the production of Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports.
5. Stop GTF and the traces after a few minutes.
6. Produce the following Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports for analysis:
   - The Accounting Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)
   - Statistics Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)
   - Locking Report Set (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)
7. Use the DB2 performance trace selectively for detailed problem analysis.

PSPI

Related concepts:

Lock contention

Types of DB2 traces

Related tasks:

Monitoring concurrency and locks

Improving concurrency

Minimizing the volume of DB2 trace data

Related reference:

Report Reference (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related information:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

Isolating resources that cause suspensions

When lock suspensions are unacceptably long or timeouts occur, you can use the DB2 performance trace for locking and the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS reports to isolate the resource that causes the suspensions.

About this task

The lockout report identifies the resources involved. The lockout trace indicates what contending process (agent) caused the timeout.

Procedure

PSPI To isolate the resources that cause a suspension:

1. Use the lockout report to identify the resources involved in the suspensions. The following figure shows a sample Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lockout report. The sample report shows that the PARALLEL plancontends with the DSNESPRR plan. It also shows that contention is occurring on partition 1 of the PARADABA.TAB1TS table space :

```
PRIMAUTH           --- L O C K   R E S O U R C E ---                      --------------- A G E N T S --------------
 PLANNAME          TYPE      NAME                    TIMEOUTS DEADLOCKS   MEMBER    PLANNAME  CONNECT   CORRID         HOLDER WAITER
------------------ --------- ----------------------- -------- ---------   --------  --------- --------  ------------   ------ ------
FPB
 PARALLEL          PARTITION DB  =PARADABA                  2         0   N/P       DSNESPRR  TSO       EOA                 2      0
                             OB  =TAB1TS
                             PART= 1
                   ** LOCKOUTS FOR PARALLEL   **            2         0
```

Figure 57. Portion of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lockout report

2. Use the lockout trace to identify the contending process (agent) that cause the timeout. The lockout trace contains information about contention from a single DB2 trace record. The following figure shows the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lockout trace. For each contender, this report shows the database object, lock state (mode), and duration for each contention for a transaction lock.

```
⋮
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE     EVENT TIMESTAMP           --- L O C K   R E S O U R C E ---
PLANNAME CONNECT               RELATED TIMESTAMP EVENT    TYPE      NAME                    EVENT SPECIFIC DATA
------------------------------ ----------------- -------- --------- ----------------------- ----------------------------------------
FPB      FPBPARAL TSO          15:25:27.23692350 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
FPB      'BLANK'  AB09C533F92E N/P                                  OB  =TAB1TS             STATE   =S          ZPARM INTERVAL=  300
PARALLEL BATCH                                                      PART= 1                 DURATION=COMMIT     INTERV.COUNTER=    1
                                                                                            HASH    =X'000020E0'
                                                                                            ------------ HOLDERS/WAITERS -----------
                                                                                            HOLDER
                                                                                            LUW='BLANK'.IPSAQ421.AB09C51F32CB
                                                                                            MEMBER  =N/P        CONNECT =TSO
                                                                                            PLANNAME=DSNESPRR   CORRID=EOA
                                                                                            DURATION=COMMIT     PRIMAUTH=KARELLE
                                                                                            STATE   =X

KARL     KARL     TSO          15:30:32.97267562 TIMEOUT  PARTITION DB  =PARADABA           REQUEST =LOCK       UNCONDITIONAL
KARL     'BLANK'  AB09C65528E6 N/P                                  OB  =TAB1TS             STATE   =IS         ZPARM INTERVAL=  300
PARALLEL TSO                                                        PART= 1                 DURATION=COMMIT     INTERV.COUNTER=    1
                                                                                            HASH    =X'000020E0'
                                                                                            ------------ HOLDERS/WAITERS -----------
                                                                                            HOLDER
                                                                                            LUW='BLANK'.IPSAQ421.AB09C51F32CB
                                                                                            MEMBER  =N/P        CONNECT =TSO
                                                                                            PLANNAME=DSNESPRR   CORRID=EOA
                                                                                            DURATION=COMMIT     PRIMAUTH=DAVE
                                                                                            STATE   =X
                                                                                            ENDUSER =DAVEUSER
                                                                                            WSNAME  =DAVEWS
                                                                                            TRANS   =DAVES TRANSACTION
LOCKING TRACE COMPLETE
```

Figure 58. Portion of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lockout trace

At this point in the investigation, you know the following information:
- The applications that contend for resources.
- The page sets for which contention occurs.
- The impact, frequency, and type of the contentions.

3. Examine the `CLASS 3 LOCK/LATCH (DB2+IRLM)` and `#OCCURRENCES` fields of theTivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report to find the suspension time and number of occurrences in the Accounting Report. The class 3 lock and latch time and number of occurrences in the accounting report are key statistics to analyze when you encounter locking problems If locking and latching are increasing the elapsed time of your transactions or batch work, investigate further. The accounting report - long shows the average elapsed times and the average number of suspensions per plan execution.
   - The class 1 average elapsed time is shown in the `ELAPSED TIME` field under `APPL(CL.1)`. The class 2 times are shown in the `ELAPSED TIME` field under `DB2(CL.2)`. that are spent in DB2. The remainder of the time is spent in the application.
   - Part of the class 2 elapsed time is spent waiting for lock or latch suspensions. This time is shown in the `(LOCK/LATCH (DB2 + IRLM)` field. Most of the suspension time is spent in IRLM LOCK+LATCH. In the Locking section, it shows that is shows that most of the suspensions are IRLM Lock Suspensions
   - The `HIGHLIGHTS` section of the report shows the number of transactions processed for the accounting interval in the `#OCCURRENCES` field. In this example DDF accounting rollup is used.

   PSPI The following figure shows a portion of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting report - long.

```
AVERAGE       APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)    CLASS 3 SUSPENSIONS   AVERAGE TIME  AV.EVENT    HIGHLIGHTS
------------  ----------  ----------  ----------    --------------------  ------------  --------    --------------------------
ELAPSED TIME    0.136869    0.022632    0.000429    LOCK/LATCH(DB2+IRLM)      0.000192      0.14    #OCCURRENCES    :  1568491
 NONNESTED      0.124535    0.010763         N/A     IRLM LOCK+LATCH          0.000188      0.04    #ALLIEDS        :        0
 STORED PROC    0.012321    0.011859         N/A     DB2 LATCH                0.000004      0.11    #ALLIEDS DISTRIB:        0
 UDF            0.000002    0.000000         N/A    SYNCHRON. I/O             0.010347      6.38    #DBATS          :  1474051
 TRIGGER        0.000010    0.000010         N/A     DATABASE I/O             0.009948      6.18    #DBATS DISTRIB. :    94440
                                                     LOG WRITE I/O            0.000399      0.19    #NO PROGRAM DATA:        0
CP CPU TIME     0.004499    0.004212    0.000333    OTHER READ I/O            0.003111      3.52    #NORMAL TERMINAT:       29
 AGENT          0.004499    0.004212         N/A    OTHER WRTE I/O            0.000002      0.00    #DDFRRSAF ROLLUP:    82834
  NONNESTED     0.002150    0.002102    0.000333    SER.TASK SWTCH            0.000291      0.05    #ABNORMAL TERMIN:        0
  STORED PRC    0.002345    0.002107         N/A     UPDATE COMMIT            0.000014      0.01    #CP/X PARALLEL. :        0
  UDF           0.000001    0.000000         N/A     OPEN/CLOSE               0.000120      0.01    #UTIL PARALLEL. :        0
  TRIGGER       0.000003    0.000003         N/A     SYSLGRNG REC             0.000004      0.00    #IO PARALLELISM :     1402
 PAR.TASKS      0.000000    0.000000         N/A     EXT/DEL/DEF              0.000015      0.00    #PCA RUP COUNT  :      N/A
                                                     OTHER SERVICE            0.000139      0.03    #RUP AUTONOM. PR:      N/A
SE CPU TIME     0.003034    0.002970         N/A    ARC.LOG(QUIES)            0.000000      0.00    #AUTONOMOUS PR  :      N/A
 NONNESTED      0.003032    0.002968         N/A    LOG READ                  0.000000      0.00    #INCREMENT. BIND:       40
 STORED PROC    0.000002    0.000002         N/A    DRAIN LOCK                0.000001      0.00    #COMMITS        :  1650053
 UDF            0.000000    0.000000         N/A    CLAIM RELEASE             0.000000      0.00    #ROLLBACKS      :     1457
 TRIGGER        0.000000    0.000000         N/A    PAGE LATCH                0.000002      0.16    #SVPT REQUESTS  :        0
                                                    NOTIFY MSGS               0.000002      0.00    #SVPT RELEASE   :        0
 PAR.TASKS      0.000000    0.000000         N/A    GLOBAL CONTENTION         0.000191      0.11    #SVPT ROLLBACK  :        0
                                                    COMMIT PH1 WRITE I/O      0.000000      0.00    MAX SQL CASC LVL:        3
SUSPEND TIME    0.000117    0.014248         N/A    ASYNCH CF REQUESTS        0.000038      0.52    UPDATE/COMMIT   :     1.54
 AGENT               N/A    0.014248         N/A    TCP/IP LOB XML            0.000071      0.05    SYNCH I/O AVG.  : 0.001623
 PAR.TASKS           N/A    0.000000         N/A    ACCELERATOR               0.000000      0.00
 STORED PROC    0.000116         N/A         N/A    AUTONOMOUS PROCEDURE           N/A       N/A
 UDF            0.000002         N/A         N/A    PQ SYNCHRONIZATION             N/A       N/A
                                                    TOTAL CLASS 3             0.014248     10.94
NOT ACCOUNT.         N/A    0.001201         N/A
DB2 ENT/EXIT         N/A        8.33         N/A
EN/EX-STPROC         N/A       46.96         N/A
EN/EX-UDF            N/A        0.00         N/A
DCAPT.DESCR.         N/A         N/A    0.000000
LOG EXTRACT.         N/A         N/A    0.000000
```

4. Use the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS locking timeout trace to obtain the information necessary to reduce overheads.

What to do next

Take action to improve concurrency for the resources involved. PSPI

Related concepts:

Suspensions and wait time

Accounting trace

Related tasks:

Monitoring locks by using statistics and accounting traces

Monitoring concurrency and locks

Resolving locking conflicts (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Related reference:

Lock monitoring with the DB2 accounting trace (DB2 Data Sharing Planning and Administration)

Accounting Long Report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Lockout Report (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Lockout Trace (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Lock suspension report

You can start DB2 performance class 6 to GTF to prepare a lock suspension report.

About this task

PSPI

Because that class 6 traces only suspensions, it does not significantly reduce performance. The following figure shows the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS lock suspension report.

```
⋮
                                                              --SUSPEND REASONS-- ---------- R E S U M E   R E A S O N S ------
PRIMAUTH      --- L O C K   R E S O U R C E --- TOTAL    LOCAL  GLOB.  S.NFY ---- NORMAL ----  TIMEOUT/CANCEL  --- DEADLOCK --
 PLANNAME     TYPE      NAME                    SUSPENDS LATCH  IRLMQ  OTHER NMBR         AET NMBR         AET NMBR        AET
------------- --------- ----------------------- -------- -----  -----  ----- ---- ----------- ---- ----------- ---- ----------
FPB
 PARALLEL     PARTITION DB  =PARADABA                  2     2      0      0    0         N/C    2  303.277805    0        N/C
                        OB  =TAB1TS                          0      0      0
                        PART= 1
LOCKING REPORT COMPLETE
```

Figure 59. Portion of the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Lock suspension report

The lock suspension report shows:

- Which plans are suspended, by plan name within primary authorization ID. For statements bound to a package, see the information about the plan that executes the package.
- What IRLM requests and which lock types are causing suspensions.
- Whether suspensions are normally resumed or end in timeouts or deadlocks.
- What the average elapsed time (AET) per suspension is.

The report also shows the reason for the suspensions, as described in Table 158.

Table 158. Reasons for suspensions

Reason	Includes
LOCAL	Contention for a local resource
LATCH	Contention for latches within IRLM (with brief suspension)
GLOB.	Contention for a global resource
IRLMQ	An IRLM queued request
S.NFY	Intersystem message sending
OTHER	Page latch or drain suspensions, suspensions because of incompatible retained locks in data sharing, or a value for service use

The preceding list shows only the first reason for a suspension. When the original reason is resolved, the request could remain suspended for a second reason.

Each suspension results in either a normal resume, a timeout, or a deadlock.

The report shows that the suspension causing the delay involves access to partition 1 of table space PARADABA.TAB1TS by plan PARALLEL. Two LOCAL suspensions time out after an average of 5 minutes, 3.278 seconds (303.278 seconds).

PSPI

Chapter 46. DB2 trace

The DB2 instrumentation facility component (IFC) provides a trace facility that you can use to record DB2 data and events.

PSPI

With the IFC, however, analysis and reporting of the trace records must take place outside of DB2. You can use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to format, print, and interpret DB2 trace output. You can view an online snapshot from trace records by using Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS or other online monitors.

Each *trace class* captures information on several subsystem events. These events are identified by many instrumentation facility component identifiers (IFCIDs). The IFCIDs are described by the comments in their mapping macros, contained in *prefix*.SDSNMACS, which is shipped to you with DB2.

PSPI

Related concepts:

DB2 trace output

Programming for the instrumentation facility interface (IFI)

Related reference:

-START TRACE (DB2) (DB2 Commands)

Related information:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

Tivoli OMEGAMON XE for Tivoli OMEGAMON XE for DB2 Performance Monitor for z/OS

Minimizing the volume of DB2 trace data

The volume of data that DB2 trace collects can be quite large. Consequently, the number of trace records that you request might impact system performance.

Procedure

PSPI

To minimize the volume of trace data:

- Specify appropriate constraints and filters in the when you start traces. By doing so, you can limit the collection of trace data to particular applications or users and to limit the data collected to particular traces and trace events. You can use *trace constraints* to limit the scope of the collected data to a particular context and to particular traces and trace events. Similarly, you can use *trace filters* to exclude the collection of trace data from specific contexts and to exclude the collection of specific traces and trace events.

  For example, you can specify constraints and filters by application and user attributes such as collection ID, package name, location name, workstation name, authorization ID, user ID, role, and more. You can also use constraints and filters to limit the collection of trace data to certain trace classes and particular trace events (IFCIDs). For a complete list of the available constraint and filter options, see -START TRACE (DB2) (DB2 Commands).

- When starting a performance trace, be sure that you know what you want to report. I/O only or SQL only, for example. See Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS for examples of which classes produce which reports. Otherwise, you might have incomplete reports and have to rerun or collect too much data, overloading the data collector.
- Use the default statistics frequency, or a smaller value. A 1 minute statistics interval is enforced for certain statistics. When the statistics trace is active, statistics are collected by SMF at all times.
- Decide whether the continuous collection of accounting data is needed. If a transaction manager provides enough accounting information, DB2 accounting might not be needed. In environments where the processor is heavily loaded, consider not running the accounting trace on a continuous basis.
- When using accounting on a continuous basis, start classes 1, 2, and 3 to SMF (SMF ACCOUNTING on installation panel DSNTIPN).
- To capture minimal package accounting information, start accounting trace classes 7 and 8. If you need more detailed accounting information for packages, such as SQL statements and locking, Start accounting trace classes 7, 8, and 10. Package accounting introduces additional CPU cost and increases the amount of SMF accounting trace records.If you need only minimal accounting information for packages, start only class 7 accounting.
- You can use the values of ACCUMACC and ACCUMUID subsystem parameters to roll up DDF and RRSAF accounting records. Rolling up records reduces the volume of the accounting data.

  Be aware, however, that using the rolled accounting records provides a trade-off. Rolling up records removes granularity from the data, which means that information about outlying transactions that perform poorly is likely to be lost in the rolled up data.

  If there is no specific problem that requires a performance analysis, start with the default ACCUMACC and ACCUMUID values, to write an accounting record for every 10 accounting intervals. If a performance issue arises for which you need detailed accounting data, update ACCUMACC to NO. When the performance problem is solved, you can set ACCUMACC back to 10. Similarly, if you find that you are generating too large a volume of accounting trace data with an ACCUMACC setting of 10, and you do not have a history of performance problems with RRSAF or distributed applications, you can increase ACCUMACC to a higher value.
- If the number of DB2 latch contentions is excessive (more than 1000 per second) in a highly CPU-constrained environment, you can turn off accounting class 3 and 8 temporarily to save some CPU resources and pursue the latch contention issues later.
- Use the performance trace only for short periods and restrict the amount of data that is collected by specifying appropriate constraints and filters in the START TRACE commands. Use the default destination GTF to allow immediate analysis of the trace information.
- Start the global trace only if a problem is under investigation, and IBM Software Support has requested a trace.

PSPI

Related tasks:

Minimizing the processing cost of DB2 traces

Controlling the collection of statistics for SQL statements

Improving DB2 log performance

Designing EDM storage space for performance

Related reference:

-START TRACE (DB2) (DB2 Commands)

Report Reference (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

DDF/RRSAF ACCUM field (ACCUMACC subsystem parameter) (DB2 Installation and Migration)

AGGREGATION FIELDS field (ACCUMUID subsystem parameter) (DB2 Installation and Migration)

Types of DB2 traces

DB2 trace can record several types of data, including: statistics, accounting, audit, performance, monitor, and global traces.

PSPI

For details on what information each IFCID returns, see the mapping macros in *prefix*.SDSNMACS.

PSPI

The trace records are written using GTF or SMF records. Trace records can also be written to storage, if you are using the monitor trace class.

Related concepts:

Diagnostic traces for attachment facilities (DB2 Administration Guide)

Related tasks:

Recording GTF trace data

Recording SMF trace data

Controlling traces (DB2 Administration Guide)

Related reference:

-START TRACE (DB2) (DB2 Commands)

-STOP TRACE (DB2) (DB2 Commands)

-MODIFY TRACE (DB2) (DB2 Commands)

-DISPLAY TRACE (DB2) (DB2 Commands)

Trace field descriptions

Trace data record format

Statistics trace

The *statistics trace* reports information about how much the DB2 system services and database services are used.

PSPI

You can use the information that the statistics trace provides to plan DB2 capacity, or to tune the entire set of active DB2 programs.

If you specify YES for the SMF STATISTICS entry in panel DSNTIPN, the statistics trace uses classes 1, 3, 4, 5, and 6 as the defaults. If the statistics trace is started using the START TRACE command, the statistics trace uses class 1 as the default.

The following table describes the DB2 statistics trace classes.

If you specify YES for the SMFSTAT subsystem parameter, the statistics trace starts automatically when you start DB2, and sends the default classes (classes 1, 3, 4, 5, and 6) statistics data to SMF. SMF records statistics data in both SMF type 100 and 102 records. IFCIDs 0001, 0002, 0202, 0225, and 0230 are of SMF type 100. All other IFCIDs in statistics trace classes are of SMF type 102.

When you specify `CLASS(7)` or `IFCID(365)` in a START TRACE or MODIFY TRACE command, DB2 writes IFCID 0365 records. These records are written to the specified destination of the statistics trace for remote locations that communicate with the subsystem. Statistics are generated only for those locations that have activity since the record was generated. The location statistics are written each time that the STATIME interval elapses. Statistics can be written for as many as 95 remote locations.

DB2 also writes statistics about all DRDA locations with the other default statistics trace data to a single SMF location named DRDA REMOTE LOCS, whenever other default statistics are written.

Statistics trace classes

The following table shows the IFCIDs that are activated for each statistics trace class.

Table 159. Classes for DB2 statistics trace

Class	Description of class	Activated IFCIDs
1	Information about system services, database statistics, statistics for the DBM1 address space, and information about the system parameters that were in effect when the trace was started. This default class is also activated when you omit the `CLASS` keyword from the START TRACE command when you start the statistics trace.	0001, 0002, 0105, 0106, 0202, 0225
2	Installation-defined statistics record	0152
3	Deadlock, lock escalation, group buffer pool, data set extension information, and indications of long-running uncommitted reads, and active log space shortages.	0172, 0196, 0250, 0258, 0261, 0262, 0313, 0330, 0337
4	DB2 exceptional conditions.	0173, 0191-0195, 0203-0210, 0235, 0236, 0238, 0267, 0268
5	DB2 data sharing statistics record.	0230
6	Storage statistics for the DB2 subsystem.	0225
7	DRDA location statistics.	0365
8	Data set I/O statistics.	0199
9	Aggregate CPU and wait time statistics by connection type.	0369

Table 159. Classes for DB2 statistics trace (continued)

Class	Description of class	Activated IFCIDs
10 - 29	Reserved.	
30 - 32	Available for local use.	

PSPI

Related tasks:

Controlling the DB2 trace (DB2 Administration Guide)

Related reference:

SMF STATISTICS field (SMFSTAT subsystem parameter) (DB2 Installation and Migration)

STATISTICS TIME field (STATIME subsystem parameter) (DB2 Installation and Migration)

DSNTIPN: Tracing parameters panel (DB2 Installation and Migration)

Trace field descriptions

Trace data record format

-START TRACE (DB2) (DB2 Commands)

-MODIFY TRACE (DB2) (DB2 Commands)

Statistics Report and Trace Blocks (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

DRDA Remote Locations (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Accounting trace

The *accounting trace* records transaction-level data that is written when the processing for a transaction is completed. It provides data that enables you to conduct DB2 capacity planning and to tune application programs.

PSPI

The DB2 accounting trace provides the following types of information:

- Start and stop times
- Number of commits and aborts
- The number of times certain SQL statements are issued
- Number of buffer pool requests
- Counts of certain locking events
- Processor resources consumed
- Thread wait times for various events
- RID pool processing
- Distributed processing
- Resource limit facility statistics

DB2 trace begins collecting these items when a thread is successfully allocated to DB2. DB2 trace writes a completed record when the thread terminates or when the authorization ID changes.

If you specified YES for SMF ACCOUNTING on installation panel DSNTIPN, the accounting trace starts automatically when you start DB2, and sends IFCIDs that are of SMF type 101 to SMF. The accounting record IFCID 0003 is of SMF type 101.

PSPI

The value of the ACCUMACC subsystem parameter controls the accumulation of accounting data by user for DDF and RRSAF threads. You can use this value to consolidate multiple accounting records.

Accounting trace classes

The following table shows the IFCIDs that are activated for each accounting trace class.

Table 160. Classes for DB2 accounting trace

Class	Description of class	Activated IFCIDs
1	Standard accounting data. This default class is also activated when you omit the CLASS keyword from the START TRACE command when you start the accounting trace.	0003, 0106, 0200, 0239
2	Entry or exit from DB2 event signaling.	0232
3	Elapsed wait time in DB2.	0006-0009, 0032, 0033, 0044, 0045, 0117, 0118, 0127, 0128, 0170, 0171, 0174, 0175, 0213-0216, 0226, 0227, 0242, 0243, 0321, 0322, 0329, 0378, 0379
4	Installation-defined accounting record.	0151
5	Time spent processing IFI requests.	0187
6	Reserved.	
7	Package-level accounting in-DB2 time.	0200, 0232, 0239, 0240
8	Package-level accounting wait time in DB2.	0006-0009, 0032, 0033, 0044, 0045, 0117, 0118, 0127, 0128, 0170, 0171, 0174, 0175, 0213-0216, 0226, 0227, 0239, 0241-0243, 0321, 0322, 0378, 0379
10	Package detail. One of the following traces must also be activated before the IFCID 0239 records are written: • Accounting class 7 • Accounting class 8 • Monitor class 7 • Monitor class 8	0239
11-29	Reserved.	
30 - 32	Available for local use.	

Several DB2 components accumulate accounting data for class 1 components during normal execution. The data is collected at the end of the accounting period, but does not involve as much overhead as individual event tracing.

When you start class 2, 3, 7, 8, or 10 many additional trace points are activated. Every occurrence of these events is traced internally by DB2 trace, but these traces are not written to any external destination. When class 2 or class 3 is activated, the accounting facility uses these traces to compute the additional total statistics that appear in the accounting record IFCID 003. Accounting class 1 must be active to externalize the information.

Classes 7 and 8 control whether IFCID 239 is externalized. One or the other must be started for IFCID 239 to be externalized.

Accounting for packages

Before you can turn on accounting for packages, accounting trace class 7 or 8 must be active. You can activate class 7 while a plan is being executed, but accounting trace information is only gathered for packages executed after class 7 is activated. Activate accounting trace class 8 to collect information about the amount of time an agent was suspended in DB2 for each executed package. Accounting class 1 determines the destination for packages accounting (IFCID239). If accounting trace classes 2 and 3 are activated, activating accounting trace classes 7 and 8 incurs minimal additional performance cost.

Classes 2 and 3

If you want information from either or both accounting class 2 and 3, be sure to activate class 2, class 3, or both classes before your application starts. If these classes are activated while the application is running, the times gathered by DB2 trace are only from the time the class was activated.

IFI class 5

Accounting trace class 5 provides information about time spent in DB2 processing instrumentation facility interface (IFI) requests.

- *Class 5 elapsed time* indicates the amount of elapsed time spent in DB2 processing instrumentation facility interface (IFI) requests. This time is included as part of the value for class 2 elapsed time.
- *Class 5 CPU time* indicates the amount of time consumed on the central processor for processing instrumentation facility interface (IFI) requests during the accounting interval. This time is a subset of and included in the values for class 2 CPU time.

If no agent issued any IFI requests, these fields are not included in the accounting record.

Related concepts:

Response times

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS accounting reports

Related tasks:

Controlling the DB2 trace (DB2 Administration Guide)

Related reference:

Trace field descriptions

Trace data record format

DSNTIPN: Tracing parameters panel (DB2 Installation and Migration)

DDF/RRSAF ACCUM field (ACCUMACC subsystem parameter) (DB2 Installation and Migration)

Audit trace

The *audit trace* collects information about DB2 security controls and can be used to ensure that data access is allowed only for authorized purposes.

PSPI

On the CREATE TABLE or ALTER TABLE statements, you can specify whether or not a table is to be audited, and in what manner; you can also audit security information such as any access denials, grants, or revokes for the table. The default causes no auditing to take place.

If you specified YES for AUDIT TRACE on installation panel DSNTIPN, audit trace class 1 starts automatically when you start DB2. By default, DB2 sends audit data to SMF. SMF records audit data in type 102 records. When you invoke the -START TRACE command, you can also specify GTF as a destination for audit data.

PSPI

The following tables shows the IFCIDs that are activated for each audit trace class.

Table 161. Classes for DB2 audit trace

Class	Description of class	Activated IFCIDs
1	Access attempts denied due to inadequate authorization. This default class is also activated when you omit the CLASS keyword from the START TRACE command when you start the audit trace.	0140
2	Explicit GRANT and REVOKE.	0141
3	CREATE, ALTER, and DROP operations against audited tables.	0142
4	First change of audited object.	0143
5	First read of audited object.	0144
6	Bind time information about SQL statements that involve audited objects.	0145
7	Assignment or change of authorization ID.	0055, 0083, 0087, 0169, 0319
8	Utilities.	0023, 0024, 0025, 0219, 0220
9	Installation-defined audit record.	146
10	Trusted context information.	0269, 0270
11	Audits of successful access.	0361[1]
12 - 29	Reserved.	
30 - 32	Available for local use.	

Notes:

1. If IFCID 0361 is started through START TRACE, all successful access is traced. If IFCID 0361 is started because audit policy category SYSADMIN is on, only successful access using the SYSADMIN administrative authority is traced. If IFCID 0361 is started because audit policy category DBADMIN is on, only successful access using the DBADMIN administrative authority is traced.

Related concepts:

Auditing access to DB2 (Managing Security)

DB2 audit trace (Managing Security)

Related tasks:

Controlling the DB2 trace (DB2 Administration Guide)

Related reference:

Audit classes (Managing Security)

Trace field descriptions

Trace data record format

DSNTIPN: Tracing parameters panel (DB2 Installation and Migration)

CREATE TABLE (DB2 SQL)

ALTER TABLE (DB2 SQL)

-START TRACE (DB2) (DB2 Commands)

Performance trace

The *performance trace* is intended for performance analysis and tuning. This trace includes records of specific events in the system, including events related to distributed data processing. The data can be used for program, resource, user, and subsystem-related tuning.

GUPI

You can use this information to further identify a suspected problem, or to tune DB2 programs and resources for individual users or for DB2 as a whole.

You cannot automatically start collecting performance data when you install or migrate DB2. The performance trace is started when you issue the following command:

```
-START TRACE(PERFM)
```

The performance trace defaults to GTF.

Table 162. Classes for DB2 performance trace

Class	Description of class	Activated IFCIDs
1	Background events. This default class is also activated when you omit the CLASS keyword from the START TRACE command when you start the performance trace.	0001, 0002, 0031, 0042, 0043, 0076-0079, 0102, 0103, 0105-0107, 0153
2	Subsystem events. This default class is also activated when you omit the CLASS keyword from the START TRACE command when you start the performance trace.	0003, 0068-0075, 0080-0089, 0106, 0174, 0175
3	SQL events. This default class is also activated when you omit the CLASS keyword from the START TRACE command when you start the performance trace.	0022, 0053, 0055, 0058-0066, 0092, 0095-0097, 0106, 0112, 0173, 0177, 0233, 0237, 0250, 0272, 0273, 0325
4	Reads to and writes from the buffer and EDM pools.	0006-0010, 0029-0030, 0105-0107, 0127, 0128, 0226, 0227, 0321, 0322

Table 162. Classes for DB2 performance trace (continued)

Class	Description of class	Activated IFCIDs
5	Write to log; archive log.	0032-0041, 0104, 0106, 0114-0120, 0228, 0229
6	Summary lock information.	0020, 0044, 0045, 0105-0107, 0172, 0196, 0213, 0214, 0218, 0337
7	Detailed lock information.	0021, 0105-0107, 0223
8	Data scanning detail.	0013-0018, 0105-0107, 0125, 0221, 0222, 0231, 0305, 0311, 0363
9	Sort detail.	0026-0028, 0095-0096, 0106
10	BIND, commands, and utilities detail.	0023-0025, 0090, 0091, 0105-0107, 0108-0111, 0201, 0256
11	Execution unit switch and latch contentions.	0046-0052, 0056, 0057, 0093, 0094, 0106, 0113
12	Storage manager.	0098-0101, 0106
13	Edit and validation exits.	0011, 0012, 0019, 0105-0107
14	Entry from and exit to an application.	0067, 0106, 0121, 0122
15	Installation-defined performance record.	0154
16	Distributed processing.	0157-0163, 0167, 0183
17	Claim and drain information.	211-216
18	Event-based console messages.	0197
19	Data set open and close activity.	0370, 0371
20	Data sharing coherency summary.	0249-0251, 0256-0257, 0261, 0262, 0267, 0268
21	Data sharing coherency detail.	0255, 0259, 0263
22	Authorization exit parameters.	0314
23	Language environment runtime diagnostics.	0327
24	Stored procedure detail.	0380, 0499
25-29	Reserved.	
30 - 32	Available for local use.	

GUPI

Related tasks:

Controlling the DB2 trace (DB2 Administration Guide)

Related reference:

Trace field descriptions

Trace data record format

DSNTIPN: Tracing parameters panel (DB2 Installation and Migration)

-START TRACE (DB2) (DB2 Commands)

Monitor trace

The *monitor trace* enables attached monitor programs to access DB2 trace data through calls to the instrumentation facility interface (IFI). Monitor programs can access the trace data asynchronously through an OP*x* buffer by issuing READA requests, or synchronously in the monitor return area by issuing READS requests.

PSPI

Monitor trace has the following predefined trace classes that are used explicitly for monitoring. The following table shows the IFCIDs activated for each monitor trace class.

Table 163. Classes for DB2 monitor types

Class	Description of class	Activated IFCIDs
1	Standard accounting data. This default class is also activated when you omit the CLASS keyword from the START TRACE command when you start the monitor trace.	0200
2	Entry or exit from DB2 event signaling. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 2 information is available in IFCID 0003 in the accounting record. Monitor class 2 is equivalent to accounting class 2 and results in equivalent overhead. Monitor class 2 times appear in IFCIDs 0147, 0148, and 0003 if either monitor trace class 2 or accounting class 2 is active.	0232
3	DB2 wait time for I/O, locks; resource usage information. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 3 information is available in the accounting record, IFCID 0003. As with monitor class 2, monitor class 3 overhead is equivalent to accounting class 3 overhead. When monitor trace class 3 is active, DB2 can calculate the duration of a class 3 event, such as when an agent is suspended due to an unavailable lock. Monitor class 3 times appear in IFCIDs 0147, 0148, and 0003, if either monitor class 3 or accounting class 3 is active.	0006-0009, 0032, 0033,0044, 0045, 0117, 0118, 0127, 0128, 0170, 0171, 0174, 0175, 0213,0 214, 0215, 0216, 0226, 0227, 0242, 0243, 0321, 0322, 0378, 0379
4	Installation-defined monitor record.	0155
5	Time spent processing IFI requests.	0187
6	Changes to tables created with DATA CAPTURE CHANGES.	0185
7	Entry or exit from DB2 event signaling for package accounting. The data traces the amount of time an agent spent in DB2 to process each package. If monitor trace class 2 is active, activating class 7 has minimal performance impact. Class 7 enables the IFCID 0239 to be externalized.	0200, 0232, 0240
8	Wait time for a package. If monitor trace class 3 is active, activating class 8 has minimal performance impact. Class 8 enables the IFCID 0239 to be externalized.	0006-0009, 0032, 0033, 0044, 0045, 0051, 0052, 0056, 0057, 0117, 0118, 0127, 0128, 0170,171,174, 175, 213-216, 226, 227, 239, 241-243, 321, 322, 0378, 0379

Table 163. Classes for DB2 monitor types (continued)

Class	Description of class	Activated IFCIDs
9	Enables statement level accounting. Provides information about statement details in IFCID 0148.	0124
10	Package detail for buffer manager, lock manager and SQL statistics. It contains the same information as accounting class 10. Monitor records do not include class 10, but it shows up in IFCID 0003 in the accounting record. Information from class 10 is written in additional sections of IFCID 0239. However, monitor class 7 or 8 must be activated for IFCID 0239 to be written. One of the following traces must also be activated before the IFCID 0239 records are written: • Accounting class 7 • Accounting class 8 • Monitor class 7 • Monitor class 8	0239
11-28	Reserved.	
29	Controls the subsystem-wide collection of statistics for SQL statements. When monitor class 29 is activated, trace records are written for the following events: • When dynamic statements are removed from the statement cache. (IFCID 0316) • When static statements are removed from the EDM pool. (IFCID 0401)	0316, 0318, 0400, 0401
30 - 32	Available for local use.	

PSPI

Related concepts:

Programming for the instrumentation facility interface (IFI)

Related tasks:

Controlling the DB2 trace (DB2 Administration Guide)

Related reference:

-START TRACE (DB2) (DB2 Commands)

Trace field descriptions

Trace data record format

DSNTIPN: Tracing parameters panel (DB2 Installation and Migration)

Recording SMF trace data

Each location is responsible for processing the SMF records produced by DB2 trace.

Procedure

PSPI

To record SMF trace data, use the following approaches:

- Use the z/OS operator command SETSMF or SS to alter SMF parameters that you specified previously.

  ```
  SETSMF SYS(TYPE(100:102))
  ```

  To execute this command, specify PROMPT(ALL) or PROMPT(LIST) in the SMFPRM*xx* member used from SYS1.PARMLIB. If you are not using measured usage licensing, do not specify type 89 records or you incur the overhead of collecting that data. Statistics (record type 100), accounting (record type 101), and performance (record type 102) data are recorded to SMF.
- Use the SMF program IFASMFDP to dump these records to a sequential data set. You might want to develop an application or use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to process these records.

PSPI

Related reference:

SMF writer header section

z/OS MVS System Management Facilities (SMF)

Activating SMF

SMF must be running before you can send data to it.

Procedure

PSPI

To make SMF operational:

- Specify the ACTIVE parameter and the proper TYPE subparameter for SYS and SUBSYS to update member SMFPRM*xx* of SYS1.PARMLIB. member SMFPRM*xx* indicates whether SMF is active and which types of records SMF accepts. For member SMFPRM*xx*, *xx* are two user-defined alphanumeric characters appended to 'SMFPRM' to form the name of an SMFPRM*xx* member.
- Optional: You can also code an IEFU84 SMF exit to process the records that are produced.

PSPI

Allocating SMF buffers

When you specify a performance trace type, the volume of data that DB2 can collect can be quite large. If you are sending this data to SMF, you must allocate adequate SMF buffers; the default buffer settings are probably insufficient.

About this task

PSPI

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. DB2 sends a message (DSNW133I) to the MVS operator when this occurs. DB2 treats the error as temporary and remains active even though data could be lost. DB2 sends another message (DSNW123I) to the z/OS operator when the shortage has been alleviated and trace recording has resumed.

You can determine if trace data has been lost by examining the DB2 statistics records with an IFCID of 0001, as mapped by macro DSNDQWST. These records show:
- The number of trace records successfully written
- The number of trace records that could not be written
- The reason for the failure

If your location uses SMF for performance data or global trace data, be sure that:
- Your SMF data sets are large enough to hold the data.
- SMF is set up to accept record type 102. (Specify member SMFPRM*xx*, for which 'xx' are two user-defined alphanumeric characters.)
- Your SMF buffers are large enough.

Procedure

To allocate SMF buffers:
- Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Do not use the default settings if DB2 performance or global trace data is sent to SMF.
- Specify CISZ(4096) and BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set. These values are the minimum required for DB2; you might have to increase them, depending on your z/OS environment.

  DB2 runs above the 16MB line of virtual storage in a cross-memory environment.

  PSPI

Reporting data in SMF

You can use several methods to see reports for SMF trace records.

About this task

PSPI

By using any of the following tools, you can compare any report for a current day, week, or month with an equivalent sample, as far back as you want to go. The samples become more widely spaced but are still available for analysis.

Procedure

To send reporting data to SMF:
- Use Tivoli Decision Support for z/OS to collect the data and create graphical or tabular reports.
- Write an application program to read and report information from the SMF data set. You can tailor it to fit your exact needs.
- Use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS DB2 Performance Expert on z/OS.

PSPI

Related reference:

z/OS MVS System Management Facilities (SMF)

Related information:

Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

Tivoli Decision Support for z/OS

Recording GTF trace data

The default destination for the performance trace classes is the generalized trace facility (GTF). The z/OS operator must start GTF before you can send data to it.

Before you begin

PSPI

Ensure that no GTF member exists in SYS1.

About this task

When starting GTF, if you use the JOBNAMEP option to obtain only those trace records written for a specific job, trace records written for other agents are not written to the GTF data set. This means that a trace record that is written by a system agent that is processing for an allied agent is discarded if the JOBNAMEP option is used. For example, after a DB2 system agent performs an IDENTIFY request for an allied agent, an IFCID record is written. If the JOBNAMEP keyword is used to collect trace data for a specific job, however, the record for the IDENTIFY request is not written to GTF, even if the IDENTIFY request was performed for the job named on the JOBNAMEP keyword.

You can record DB2 trace data in GTF using a GTF event ID of X'FB9'.

Trace records longer than the GTF limit of 256 bytes are spanned by DB2.

Procedure

To start GTF:

Start GTF as indicated in the following table to ensure that offsets map correctly. When starting GTF, specify TIME=YES, and then TRACE=USRP.

Table 164. Recording GTF trace data

You enter...	System responds...
`S GTF,,,(TIME=YES)`	`AHL100A SPECIFY TRACE OPTIONS`
`TRACE=USRP`	`AHL101A SPECIFY TRACE EVENT KEYWORDS --USR=`
`USR=(FB9)`	`AHL102A CONTINUE TRACE DEFINITION OR REPLY END`
`END`	`AHL125A RESPECIFY TRACE OPTIONS OR REPLY U`
`U`	`AHL031I GTF INITIALIZATION COMPLETE`

To make stopping GTF easier, you can name the GTF session when you start it. For example, you could specify `S GTF.GTF,,,(TIME=YES)`

Results

If a GTF member exists in SYS1.PARMLIB, the GTF trace option USR might not be in effect. When no other member exists in SYS1.PARMLIB, you are sure to have only the USR option activated, and no other options that might add unwanted data to the GTF trace.

PSPI

Related concepts:

The Generalized Trace Facility (GTF) (MVS Diagnosis: Tools and Service Aids)

DB2 trace output

When you activate a DB2 trace, it produces trace records based on the parameters you specified for the START TRACE command.

PSPI

Each record identifies one or more significant DB2 events. You can use Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS to format, print, and interpret DB2 trace output. If you do not have Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, or you want to do your own analysis of the trace output, you can use this information and the trace field descriptions that are shipped with DB2. By examining a DB2 trace record, you can determine the type of trace that produced the record (statistics, accounting, audit, performance, monitor, or global) and the event the record reports.

Note that when the trace output indicates a particular release level, 'xx' varies according to the actual release of DB2.

PSPI

The sections of the trace output

Trace records can be written to SMF or GTF.

PSPI

In both cases, the record contains up to four basic sections:

- An SMF or GTF writer header section
- A self-defining section
- A product section
- Zero or more data sections

The following figure shows the format of DB2 trace records.

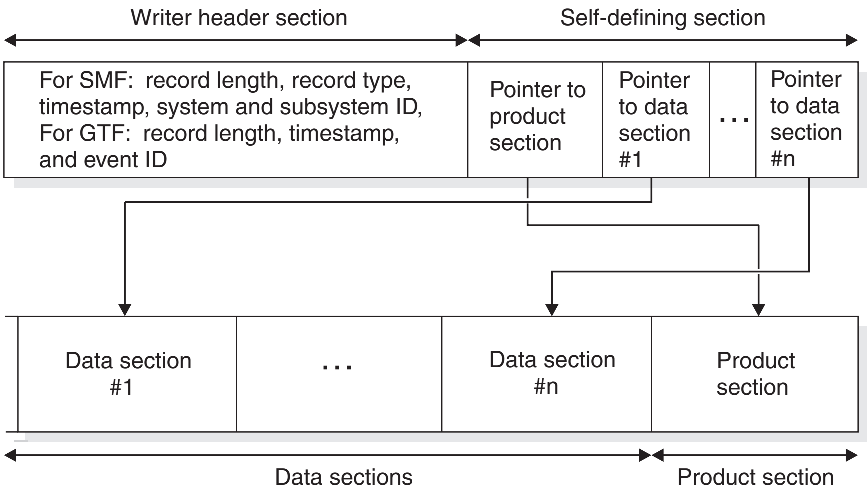

Writer header section
Self-defining section
For SMF: record length, record type, timestamp, system and subsystem ID, For GTF: record length, timestamp, and event ID
Pointer to product section
Pointer to data section #1
. . .
Pointer to data section #n
Data section #1
. . .
Data section #n
Product section
Data sections
Product section

Figure 60. General format of trace records written by DB2

The writer header section begins at the first byte of the record and continues for a fixed length. (The GTF writer header is longer than the SMF writer header.)

The self-defining section follows the writer header section (both GTF and SMF). The first self-defining section always points to a special data section called the product section. Among other things, the product section contains an instrumentation facility component identifier (IFCID). Descriptions of the records in the data section differ for each IFCID.

The product section also contains field QWHSNSDA, which indicates how many self-defining data sections the record contains. You can use this field to keep from trying to access data sections that do not exist. In trying to interpret the trace records, remember that the various keywords you specified when you started the trace determine whether any data is collected. If no data has been collected, field QWHSNSDA shows a data length of zero.

PSPI

Related concepts:

Types of DB2 traces

The Generalized Trace Facility (GTF) (MVS Diagnosis: Tools and Service Aids)

Related reference:

-START TRACE (DB2) (DB2 Commands)

z/OS MVS System Management Facilities (SMF)

SMF writer header section

In SMF, writer headers for statistics records are mapped by macro DSNDQWST, for accounting records by DSNDQWAS, and for performance, audit, and monitor records by DSNDQWSP.

PSPI

When these macros are assembled, they include the other macros necessary to map the remainder of the trace records sent to SMF.

The SMF writer header section begins at the first byte of the record. After establishing addressability, you can examine the header fields. The fields are described in Table 165.

Table 165. Contents of SMF writer header section

Hex Offset	DSNDQWST	DSNDQWAS	DSNDQWSP	Description
0	SM100LEN	SM101LEN	SM102LEN	Total length of SMF record
2	SM100SGD	SM101SGD	SM102SGD	Segment descriptor
4	SM100FLG	SM101FLG	SM102FLG	System indicator
5	SM100RTY	SM101RTY	SM102RTY	SMF record type: • Statistics=100(dec) • Accounting=101(dec) • Monitor=102(dec) • Audit=102(dec) • Performance=102(dec)
6	SM100TME	SM101TME	SM102TME	SMF record timestamp, time portion
A	SM100DTE	SM101DTE	SM102DTE	SMF record timestamp, date portion
E	SM100SID	SM101SID	SM102SID	System ID
12	SM100SSI	SM101SSI	SM102SSI	Subsystem ID
16	SM100STF	SM101STF	SM102STF	Reserved
17	SM100RI	SM101RI	SM102RI	Reserved
18	SM100SEQ	SM101SQ	SM102SEQ	Compression information, which consists of the next two fields.
18	SM100CMP	SM101CMP	SM101CMP	High-order bit value: **1** The record is compressed. **0** The record is uncompressed.
18	SM100BUF	SM101BUF	SM102BUF	The rest of the field. If the high-order bit is 1, this value is the length of the data record when the record is uncompressed. If the high-order bit is 0, this value is 0.
1C	SM100END	SM101END	SM102END	End of SMF header

Figure 61 is a sample of the first record of the DB2 performance trace output sent to SMF.

```
          A         B   C       D        E          F                     G   H
000000   01240000 0E660030  9EEC0093 018FF3F0   F9F0E2E2 D6D70000  00000000 0000008C
         I    J    K        L   M    N
000020   00980001 0000002C  005D0001 00550053   4DE2E3C1 D9E340E3  D9C1C3C5 404DE2E3
000040   C1E3405D C3D3C1E2  E2404D5C 405DD9D4   C9C4404D 5C405DD7  D3C1D540 4D5C405D
000060   C1E4E3C8 C9C4404D  5C405DC9 C6C3C9C4   404D5C40 5DC2E4C6  E2C9E9C5 404D5C40
                                     O          P   Q  R
000080   5D000000 01000101  01000000 004C0110   000402xx 00B3AB78  E2E2D6D7 A6E9BACB
                                                S
0000A0   F6485E02 00000003  00000021 00000001   E2C1D5E3 C16DE3C5  D9C5E2C1 6DD3C1C2
0000C0   C4C2F2D5 C5E34040  D3E4D5C4 F0404040   A6E9BACB F4570001  004C0200 E2E8E2D6
0000E0   D7D94040 F0F2F34B  C7C3E2C3 D5F6F0F2   E2E2D6D7 40404040  40404040 40404040
000100   E2E8E2D6 D7D94040  00000000 00000000   00000000 00000000  00000000 00000000
000120   00000000T
```

Figure 61. DB2 trace output sent to SMF (printed with DFSERA10 print program of IMS)

Key to Figure 61 on page 802	Description
A 0124	Record length (field SM102LEN); beginning of SMF writer header section
B 66	Record type (field SM102RTY)
C 0030 9EEC	Time (field SM102TME)
D 0093 018F	Date (field SM102DTE)
E F3F0 F9F0	System ID (field SM102SID)
F E2E2 D6D7	Subsystem ID (field SM102SSI)
G	End of SMF writer header section
H 0000008C	Offset to product section; beginning of self-defining section
I 0098	Length of product section
J 0001	Number of times the product section is repeated
K 0000002C	Offset to first (in this case, only) data section
L 005D	Length of data section
M 0001	Number of times the data section is repeated
N 00550053	Beginning of data section
O	Beginning of product section
P 0004	IFCID (field QWHSIID)
Q 02	Number of self-defining sections in the record (field QWHSNSDA)
R xx	Release indicator number (field QWHSRN); this varies according to the actual level of DB2 you are using.
S E2C1D5E3...	Local location name (16 bytes)
T	End of first record

PSPI

GTF writer header section

The length and content of the writer header section differs between SMF and GTF records, but the other sections of the records are the same for SMF and GTF.

PSPI

The GTF writer header section begins at the first byte of the record. After establishing addressability, you can examine the fields of the header. The writer headers for trace records sent to GTF are always mapped by macro DSNDQWGT. The fields are described in Table 166.

Table 166. Contents of GTF writer header section

Offset	Macro DSNDQWGT field	Description
0	QWGTLEN	Length of Record
2		Reserved
4	QWGTAID	Application identifier
5	QWGTFID	Format ID

Table 166. Contents of GTF writer header section (continued)

Offset	Macro DSNDQWGT field	Description
6	QWGTTIME	Timestamp; you must specify TIME=YES when you start GTF.
14	QWGTEID	Event ID: X'EFB9'
16	QWGTASCB	ASCB address
20	QWGTJOBN	Job name
28	QWGTHDRE	Extension to header
28	QWGTDLEN	Length of data section
30	QWGTDSCC	Segment control code 0=Complete 2=Last 1=First 3=Middle
31	QWGTDZZ2	Reserved
32	QWGTSSID	Subsystem ID
36	QWGTWSEQ	Sequence number
40	QWGTEND	End of GTF header

Figure 62 on page 805 contains trace output sent to GTF.

```
DFSERA10 - PRINT PROGRAM

 000000   001A0000 0001FFFF  94B6A6E9 BD6636FA   5C021000 00010000  0000
          A            B                  C               D                  E   F
 000000   011C0000 FF00A6E9  C33E28F7 DD03EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000100
          G               H   I        J    K      L         M    N      O
 000020   E2E2D6D7 00000001  000000A0 00980001   00000038 00680001  0060005E 4DE2E3C1
 000040   D9E340E3 D9C1C3C5  404DE2E3 C1E3405D   C3D3C1E2 E2404D5C  405DD9D4 C9C4404D
 000060   5C405DC4 C5E2E340  4DC7E3C6 405DD7D3   C1D5404D 5C405DC1  E4E3C8C9 C4404D5C
 000080   405DC9C6 C3C9C440  4D5C405D C2E4C6E2   C9E9C540 4D5C405D  FFFFFFFF 00040101
          P         Q   R  S
 0000A0   004C0110 000402xx  00B3ADB8 E2E2D6D7   A6E9C33E 28EF4403  00000006 00000001
                   T
 0000C0   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4
 0000E0   F0404040 A6E9C33E  271F0001 004C0200   E2E8E2D6 D7D94040  F0F2F34B C7C3E2C3
 000100   D5F6F0F2 E2E2D6D7  40404040 40404040   40404040 E2E8E2D6  D7D94040
                                                                                 U
 000000   00440000 FF00A6E9  C33E2901 1303EFB9   00F91400 E2E2D6D7  D4E2E3D9 00280200
 000020   E2E2D6D7 00000001  00000000 00000000   00000000 00000000  00000000 00000000
 000040   00000000 V
          W                                                                       X
 000000   011C0000 FF00A6E9  C33E2948 E203EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000100
 000020   E2E2D6D7 00000002  000006D8 004C0001   00000090 001C0004  00000100 001C000E
 000040   00000288 0018000E  00000590 00400001   000005D0 00740001  00000480 00440001
 000060   000003D8 00800001  00000458 00280001   00000644 00480001  000004E4 00AC0001
 000080   0000068C 004C0001  000004C4 00200001   D4E2E3D9 00000001  762236F2 00000000
 0000A0   59F48900 001E001E  00F91400 C4C2D4F1   00000001 1A789573  00000000 95826100
 0000C0   001F001F 00F90E00  C4C9E2E3 00000000   3413C60E 00000000  1C4D0A00 00220022
 0000E0   00F90480 C9D9D3D4  00000000 0629E2BC   00000000 145CE000  001D001D 00F91600
 000100   E2D4C640 00000046  00000046 00000000   00000000 00000000  00000000
                                                                                 Y
 000000   011C0000 FF00A6E9  C33E294B 1603EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
 000020   E2E2D6D7 00000002  D9C5E240 00000000   00000000 00000000  00000000 00000000
 000040   00000000 C7E3C640  00000001 00000001   00000000 00000000  00000000 00000000
 000060   E2D9E540 00000000  00000000 00000000   00000000 00000000  00000000 E2D9F140
 000080   00000156 000000D2  00000036 00000036   00000000 00000004  E2D9F240 00000000
 0000A0   00000000 00000000  00000000 00000000   00000000 D6D7F140  00000000 00000000
 0000C0   00000000 00000000  00000000 00000000   D6D7F240 00000000  00000000 00000000
 0000E0   00000000 00000000  00000000 D6D7F340   00000000 00000000  00000000 00000000
 000100   00000000 00000000  D6D7F440 00000000   00000000 00000000  00000000
                                                                                 Y
```

```
000000   011C0000 FF00A6E9  C33E294D 3C03EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  00000000 00000000   D6D7F540 00000000  00000000 00000000
000040   00000000 00000000  00000000 D6D7F640   00000000 00000000  00000000 00000000
000060   00000000 00000000  D6D7F740 00000000   00000000 00000000  00000000 00000000
000080   00000000 D6D7F840  00000000 00000000   00000000 00000000  00000000 00000000
0000A0   00010000 0000000E  0000000D 00000000   00000000 00000000  00020000 0000000D
0000C0   0000000D 00000000  00000000 00000000   00030000 00000003  00000003 00000000
0000E0   00000000 00000000  00040000 00000006   00000006 00000000  00000000 00000000
000100   00050000 00000005  00000005 00000000   00000000 00000000  006A0000
                                                                               Y
000000   011C0000 FF00A6E9  C33E294F 6103EFB9   00F91400 E2E2D6D7  D4E2E3D9 01000300
000020   E2E2D6D7 00000002  00000005 00000005   00000000 00000000  00000000 008C0000
000040   00000000 00000000  00000000 00000000   00000000 008D0000  00000000 00000000

⋮
                                                                               Z
000000   00780000 FF00A6E9  C33E2957 D103EFB9   00F91400 E2E2D6D7  D4E2E3D9 005C0200
                                     AA
000020   E2E2D6D7 00000002  00000000 004C011A   00010D31 02523038  E2E2D6D7 A6E9C33E
000040   29469A03 0000000E  00000002 00000001   E2C1D5E3 C16DE3C5  D9C5E2C1 6DD3C1C2
000060   40404040 40404040  40404040 40404040   A6E9B6B4 9A2B0001
```

Figure 62. DB2 trace output sent to GTF (spanned records printed with DFSERA10 print program of IMS)

Key to Figure 62	Description
A 011C	Record length (field QWGTLEN); beginning of GTF writer header section
B A6E9 C33E28F7 DD03	Timestamp (field QWGTTIME)
C EFB9	Event ID (field QWGTEID)
D E2E2D6D7 D4E2E3D9	Job name (field QWGTJOBN)
E 0100	Length of data section
F 01	Segment control code (01 = first segment of the first record)
G E2E2D6D7	Subsystem ID (field QWGTSSID)
H	End of GTF writer header section
I 000000A0	Offset to product section; beginning of self-defining section
J 0098	Length of product section
K 0001	Number of times the product section is repeated
L 00000038	Offset to first (in this case, only) data section
M 0068	Length of data section
N 0001	Number of times the data section is repeated
O 0060005E	Beginning of data section
P 004C0110...	Beginning of product section
Q 0004	IFCID (field QWHSIID)
R 02	Number of self-defining sections in the record (field QWHSNSDA)
S xx	Release indicator number (field QWHSRN); this varies according to the actual release level of DB2 you are using.
T E2C1D5E3...	Local location name (16 bytes)
U 02	Last segment of the first record
V	End of first record

Key to Figure 62 on page 805	Description
W	Beginning of GTF header for new record
X 01	First segment of a spanned record (QWGTDSCC = QWGTDS01)
Y 03	Middle segment of a spanned record (QWGTDSCC = QWGTDS03)
Z 02	Last segment of a spanned record (QWGTDSCC = QWGTDS02)
AA 004C	Beginning of product section

GTF records are blocked to 256 bytes. Because some of the trace records exceed the GTF limit of 256 bytes, they have been blocked by DB2. Use the following logic to process GTF records:

1. Is the GTF event ID of the record equal to the DB2 ID (that is, does QWGTEID = X'*x*FB9')?

   If it is *not* equal, get another record.

   If it is equal, continue processing.
2. Is the record spanned?

   If it is spanned (that is, QWGTDSCC ¬ = QWGTDS00), test to determine whether it is the first, middle, or last segment of the spanned record.

   a. If it is the *first* segment (that is, QWGTDSCC = QWGTDS01), save the entire record including the sequence number (QWGTWSEQ) and the subsystem ID (QWGTSSID).
   b. If it is a *middle* segment (that is, QWGTDSCC = QWGTDS03), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWTGSSID). Then move the data portion immediately after the GTF header to the end of the previous segment.
   c. If it is the *last* segment (that is, QWGTDSCC = QWGTDS02), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWTGSSID). Then move the data portion immediately after the GTF header to the end of the previous record.

      Now process the completed record.

   If it is **not** spanned, process the record.

The following figure shows the same output after it has been processed by a user-written routine, which follows the logic that was outlined previously.

```
000000   01380000 FF00A6E9  DCA7E275 1204EFB9   00F91400 E2E2D6D7  D4E2E3D9 011C0000
000020   E2E2D6D7 00000019  000000A0 00980001   00000038 00680001  0060005E 4DE2E3C1
000040   D9E340E3 D9C1C3C5  404DE2E3 C1E3405D   C3D3C1E2 E2404D5C  405DD9D4 C9C4404D
000060   5C405DC4 C5E2E340  4DC7E3C6 405DD7D3   C1D5404D 5C405DC1  E4E3C8C9 C4404D5C
000080   405DC9C6 C3C9C440  4D5C405D C2E4C6E2   C9E9C540 4D5C405D  00000001 00040101
0000A0   004C0110 000402xx  00B3ADB8 E2E2D6D7   0093018F 11223310  0000000C 00000019
0000C0   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4
0000E0   F0404040 A6E9DCA7  DF960001 004C0200   E2E8E2D6 D7D94040  F0F2F34B C7C3E2C3
000100   D5F6F0F2 E2E2D6D7  40404040 40404040   40404040 E2E8E2D6  D7D94040 00000000
000120   00000000 00000000  00000000 00000000   00000000 00000000
         A                              B
000000   07240000 FF00A6E9  DCA8060C 2803EFB9   00F91400 E2E2D6D7  D4E2E3D9 07080000
                         C  D                   E
000020   E2E2D6D7 0000001A  000006D8 004C0001   00000090 001C0004  00000100 001C000E
000040   00000288 0018000E  00000590 00400001   000005D0 00740001  00000480 00440001
000060   000003D8 00800001  00000458 00280001   00000644 00480001  000004E4 00AC0001
                            F
000080   0000068C 004C0001  000004C4 00200001   D4E2E3D9 00000003  27BCFDBC 00000000
0000A0   AB000300 001E001E  00F91400 C4C2D4F1   00000001 1DE8AEE2  00000000 DB0CB200
0000C0   001F001F 00F90E00  C4C9E2E3 00000000   4928674B 00000000  217F6000 00220022
0000E0   00F90480 C9D9D3D4  00000000 07165F79   00000000 3C2EF500  001D001D 00F91600
000100   E2D4C640 0000004D  0000004D 00000000   00000000 00000000  00000000 D9C5E240
000120   00000000 00000000  00000000 00000000   00000000 00000000  C7E3C640 00000019
000140   00000019 00000000  00000000 00000000   00000000 E2D9E540  00000000 00000000
000160   00000000 00000000  00000000 00000000   E2D9F140 00000156  000000D2 00000036
000180   00000036 00000000  00000004 E2D9F240   00000092 00000001  00000091 00000091
0001A0   00000000 0000000C  D6D7F140 00000002   00000001 00000001  00000000 00010000
0001C0   20000004 D6D7F240  00000000 00000000   00000000 00000000  00000000 00000000
0001E0   D6D7F340 00000000  00000000 00000000   00000000 00000000  00000000 D6D7F440
000200   00000000 00000000  00000000 00000000   00000000 00000000  D6D7F540 00000000
000220   00000000 00000000  00000000 00000000   00000000 D6D7F640  00000000 00000000
000240   00000000 00000000  00000000 00000000   D6D7F740 00000000  00000000 00000000
000260   00000000 00000000  00000000 D6D7F840   00000000 00000000  00000000 00000000
000280   00000000 00000000  00010000 00000042   00000011 00000030  00000000 00000000
0002A0   00020000 00000041  00000011 00000030   00000000 00000000  00030000 00000003
0002C0   00000003 00000000  00000000 00000000   00040000 0000000C  0000000C 00000000
0002E0   00000000 00000000  00050000 0000000B   0000000A 00000001  00000000 00000000
000300   006A0000 0000000C  0000000B 00000001   00000000 00000000  008C0000 00000000
000320   00000000 00000000  00000000 00000000   008D0000 00000000  00000000 00000000
000340   00000000 00000000  008E0000 00000000   00000000 00000000  00000000 00000000
000360   008F0000 00000000  00000000 00000000   00000000 00000000  00900000 00000000
```

Figure 63. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS)

```
000380   00000000 00000000  00000000 00000000   00910000 00000000  00000000 00000000
0003A0   00000000 00000000  00920000 00000000   00000000 00000000  00000000 00000000
0003C0   00CA0000 00000041  00000011 00000030   00000000 00000000  00000000 00000000
0003E0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000400   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000420   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000440   00000000 00000000  00000000 00000004   00000000 00000000  000005D4 00000130
000460   0000000D 0000000A  00000029 00000009   000000C3 00000000  00000000 00000000
000480   00000001 0000000C  00000000 04A29740   00000000 00000000  00000001 00000000
0004A0   00000001 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004C0   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
0004E0   00000000 E2C1D56D  D1D6E2C5 40404040   40404040 00000000  00000002 00000003
000500   00000000 000004A8  000005C7 00000000   00000001 00000003  00000003 00000000
000520   00000001 00000000  00000001 00000000   00000000 00000000  00000000 00000000
000540   00000002 00000001  00000000 00000000   00000000 00000000  00000000 00000000
000560   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000580   00000000 00000000  00000002 00000000   00000003 00000000  00000003 00000006
0005A0   00000000 00000000  00000000 00000000   00000005 00000003  00000000 00000000
0005C0   00000000 00000003  00000000 00000000   00000000 00000000  00000000 00000000
0005E0   00000000 00000000  0000000C 00000001   00000000 00000007  00000000 00000000
000600   00000000 00000000  00000000 00000001   00000000 00000000  00000000 00000000
000620   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000640   00000000 003C0048  D8E2E2E3 00000035   00000006 00000002  0000009E 0000002B
000660   00000078 00000042  00000048 000000EE   0000001B 0000007B  0000004B 00000000
000680   00000000 00000000  00000000 0093004C   D8D1E2E3 00000000  000000FC 0000000E
0006A0   00000000 00000000  0000009D 00000000   00000000 00000016  0000000F 00000018
                                                                   G
0006C0   00000000 00000000  00000000 00000000   00000000 00000000  004C011A 00010Dxx
0006E0   02523038 E2E2D6D7  0093018F 11223324   00000042 0000001A  00000001 E2C1D5E3
000700   C16DE3C5 D9C5E2C1  6DD3C1C2 40404040   40404040 40404040  40404040 A6E9B6B4
000720   9A2B0001 H
```

Figure 64. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS) continued

Key to Figure 63 on page 807	Description
A 0724	Length of assembled record; beginning of GTF writer header section of second record (field QWGTLEN)
B EFB9	GTF event ID (field QWGTEID)
C	End of GTF writer header section of second record
D 000006D8	Offset to product section
E 00000090	Offset to first data section
F 000004C4	Offset to last data section
G 004C011A	Beginning of product section
H	End of second record

PSPI

Self-defining section

The self-defining section, which follows the writer header, contains pointers that enable you to find the product and data sections, which contain the actual trace data.

PSPI

Each "pointer" is a descriptor that contains fields, which are:

- A fullword that contains the offset from the beginning of the record to the data section.

- A halfword that contains the length of each item in the data section. If this value is zero, the length of the items are varied.
- A halfword that contains the number of times that the data section is repeated. If the field contains “0”, the data section is not in the record. If it contains a number greater than 1, multiple data items are stored contiguously within that data section. To find the second data item, add the length of the first data item to the address of the first data item (and so forth).

Pointers occur in a fixed order, and their meanings are determined by the IFCID of the record. Different sets of pointers can occur, and each set is described by a separate DSECT. Therefore, to examine the pointers, you must first establish addressability by using the DSECT that provides the appropriate description of the self-defining section. To do this, perform the following steps:

1. Compute the address of the self-defining section.

   The self-defining section begins at label “SM100END” for statistics records, “SM101END” for accounting records, and “SM102END” for performance and audit records. It does not matter which mapping DSECT you use because the length of the SMF writer header is always the same.

   For GTF, use QWGTEND.
2. Determine the IFCID of the record.

   Use the first field in the self-defining section; it contains the offset from the beginning of the record to the product section. The product section contains the IFCID.

   The product section is mapped by DSNDQWHS; the IFCID is mapped by QWHSIID.

   For statistics records that have IFCID 0001, establish addressability using label “QWS0”; for statistics records having IFCID 0002, establish addressability using label “QWS1”. For accounting records, establish addressability using label “QWA0”. For performance and audit records, establish addressability using label “QWT0”.

After establishing addressability using the appropriate DSECT, use the pointers in the self-defining section to locate the record's data sections.

PSPI

Reading the self-defining section for same-length data items:

If the length of each item in a data section is the same, the self-defining section indicates the length in a halfword that follows the fullword offset. If this value is “0”, the length of the data items varies.

PSPI

For more information about variable-length data items, see Reading the self-defining section for variable-length data items.

The relationship between the contents of the self-defining section “pointers” and the items in a data section for same-length data items is shown in The figure below.

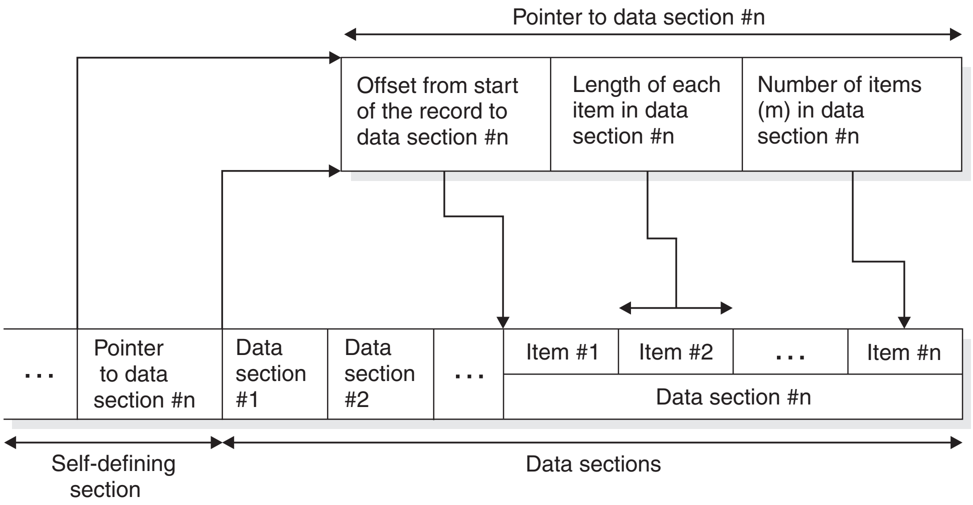

Pointer to data section #n
Offset from start of the record to data section #n
Length of each item in data section #n
Number of items (m) in data section #n
. . .
Pointer to data section #n
Data section #1
Data section #2
. . .
Item #1
Item #2
. . .
Item #n
Data section #n
Self-defining section
Data sections

Figure 65. Relationship between self-defining section and data sections for same-length data items

PSPI

Reading the self-defining section for variable-length data items:

If the length of each item in a data section varies, the self-defining section indicates the value "0" in the halfword that follows the fullword offset. The length of each variable-length data item is indicated by two bytes that precede each data item.

PSPI

The relationship between the contents of the self-defining section "pointers" and the items in a data section for variable-length data items is shown in the following figure.

PSPI

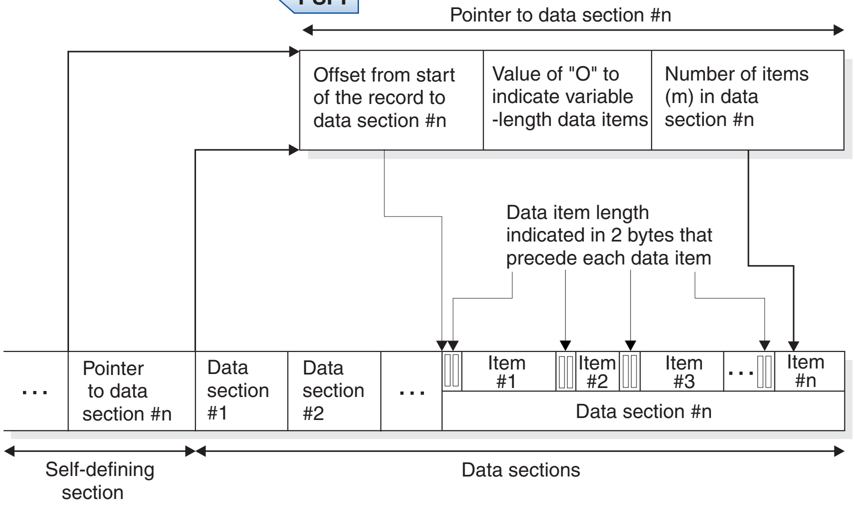

Pointer to data section #n
Offset from start of the record to data section #n
Value of "O" to indicate variable -length data items
Number of items (m) in data section #n
Data item length indicated in 2 bytes that precede each data item
. . .
Pointer to data section #n
Data section #1
Data section #2
. . .
Item #1
Item #2
Item #3
. . .
Item #n
Data section #n
Self-defining section
Data sections

Figure 66. Relationship between self-defining section and data sections for variable-length data items

Product section

The product section for all record types contains the standard header. The other headers (correlation, CPU, distributed, and data sharing data) might also be present.

PSPI

The following table shows the contents of the product section standard header.

Table 167. Contents of product section standard header

Hex Offset	Macro DSNDQWHS field	Description
0	QWHSLEN	Length of standard header
2	QWHSTYP	Header type
3	QWHSRMID	RMID
4	QWHSIID	IFCID
6	QWHSRELN	Release number section
6	QWHSNSDA	Number of self-defining sections
7	QWHSRN	DB2release identifier
8	QWHSACE	ACE address
C	QWHSSSID	Subsystem ID
10	QWHSSTCK	Timestamp—STORE CLOCK value assigned by DB2
18	QWHSISEQ	IFCID sequence number
1C	QWHSWSEQ	Destination sequence number
20	QWHSMTN	Active trace number mask
24	QWHSLOCN	Local location Name
34	QWHSLWID	Logical unit of work ID
34	QWHSNID	Network ID
3C	QWHSLUNM	LU name
44	QWHSLUUV	Uniqueness value
4A	QWHSLUCC	Commit count
4C	QWHSFLAG	Flags
4E	QWHSLOCN_Off	If QWHSLOCN is truncated, this is the offset from the beginning of QWHS to QWHSLOCN_LEN. If the value is zero, refer to QWHSLOCN.
Defined by QWHSLOCN_Off	QWHSLOCN_D	This field contains both QWHSLOCN_Len and QWHSLOCN_Var. This element is only present if QWHSLOCN_Off is greater than 0.
Defined by QWHSLOCN_Off	QWHSLOCN_Len	The length of the field. This element is only present if QWHSLOCN_Off is greater than 0.
Defined by QWHSLOCN_Off	QWHSLOCN_Var	The local location name. This element is only present if QWHSLOCN_Off is greater than 0.
50	QWHSSUBV	The sub-version for the base release.
52	QWHSEND	End of product section standard header

The following table shows the contents of the product section correlation header.

Table 168. Contents of product section correlation header

Hex Offset	Macro DSNDQWHC field	Description
0	QWHCLEN	Length of correlation header
2	QWHCTYP	Header type
3		Reserved
4	QWHCAID	Authorization ID
C	QWHCCV	Correlation ID
18	QWHCCN	Connection name
20	QWHCPLAN	Plan name
28	QWHCOPID	Original operator ID
30	QWHCATYP	The type of system that is connecting
34	QWHCTOKN	Trace accounting token field
4A		Reserved
4C	QWHCEUID	User ID at the workstation for the user
5C	QWHCEUTX	Transaction name for the user
7C	QWHCEUWN	Workstation name for the user
8E	QWHCAID_Off	If QWHCAID is truncated, this is the offset from the beginning of QWHC to QWHCAID_LEN. If the value is zero, refer to QWHCAID.
Defined by QWHCAID_Off	QWHCAID_D	This field contains both QWHCAID_Len and QWHCAID_Var.
Defined by QWHCAID_Off	QWHCAID_Len	Length of the field
Defined by QWHCAID_Off	QWHCAID_Var	Authorization ID
90	QWHCOPID_Off	If QWHCOPID is truncated, this is the offset from the beginning of QWHC to QWHCAID_LEN. If the value is zero, refer to QWHCOPID.
Defined by QWHCOPID_Off	QWHCOPID_D	This field contains both QWHCOPID_Len and QWHCOPID_Var.
Defined by QWHCOPID_Off	QWHCOPID_Len	Length of the field
Defined by QWHCOPID_Off	QWHCOPID_Var	Original operator ID
92	QWHCEUID_Off	If QWHCEUID is truncated, this is the offset from the beginning of QWHC to QWHCEUID_LEN. If the value is zero, refer to QWHCEUID. Trusted context and role data is present if an agent running under a trusted context writes the record and the trusted context data can be accessed.
Defined by QWHCEUID_Off	QWHCEUID_D	This field contains both QWHCEUID_Len and QWHCEUID_Var.
Defined by QWHCEUID_Off	QWHCEUID_Len	Length of the field
Defined by QWHCEUID_Off	QWHCEUID_Var	User's USERID

Table 168. Contents of product section correlation header (continued)

Hex Offset	Macro DSNDQWHC field	Description
94	QWHCTCXT_Off	If QWHCTCXT is truncated, this is the offset from the beginning of QWHC to QWHCTCXT_LEN. If the value is zero, refer to QWHCTCXT.
Defined by QWHCTCXT_Off	QWHCTCXT_D	This field contains both QWHCTCXT_Len and QWHCTCXT_Var.
Defined by QWHCTCXT_Off	QWHCTCXT_Len	Length of the field
Defined by QWHCTCXT_Off	QWHCTCXT_Var	Trusted Context name
96	QWHCROLE_Off	If QWHCROLE is truncated, this is the offset from the beginning of QWHC to QWHCROLE_LEN. If the value is zero, refer to QWHCROLE.
Defined by QWHCROLE_Off	QWHCROLE_D	This field contains both QWHCROLE_Len and QWHCROLE_Var.
Defined by QWHCROLE_Off	QWHCROLE_Len	Length of the field
Defined by QWHCROLE_Off	QWHCROLE_Var	Role name associated with authid
98	QWHCOAUD_Off	Offset from QWHC to the original application USERID.
Defined by QWHCOAUD_Off	QWHCOAUD_D	This field contains both QWHCOAUD_Len and QWHCOAUD_Var.
Defined by QWHCOAUD_Off	QWHCOAUD_Len	Length of the field
Defined by QWHCOAUD_Off	QWHCOAUD_Var	Original application USERID.
9A	QWHCCTKN_Off	Offset from QWHC to the correlation token. This element is only present if QWHSSUBV is greater than 0.
Defined by QWHCCTKN_Off	QWHCCTKN_D	This field contains both QWHCCTKN_Len and QWHCCTKN_Var.
Defined by QWHCCTKN_Off	QWHCCTKN_Len	Length of the field
Defined by QWHCCTKN_Off	QWHCCTKN_Var	Correlation token.
9C	QWHCEND	End of product section correlation header

The following table shows the contents of the CPU header.

Table 169. Contents of CPU header

Hex Offset	Macro DSNDQWHU field	Description
0	QWHULEN	Length of CPU header
2	QWHUTYP	Header type
3		Reserved
4	QWHUCPU	CPU time of the currently dispatched execution unit (TCB or SRB). This time includes CPU time that was consumed on an IBM specialty engine.
C	QWHUCNT	Count field reserved

Table 169. Contents of CPU header (continued)

Hex Offset	Macro DSNDQWHU field	Description
E	QWHUEND	End of header

The following table shows the contents of the distributed data header.

Table 170. Contents of distributed data header

Hex Offset	Macro DSNDQWHD field	Description
0	QWHDLEN	Length of the distributed header
2	QWHDTYP	Header type
3		Reserved
4	QWHDRQNM	Requester location name
14	QWHDTSTP	Timestamp for DBAT trace record
1C	QWHDSVNM	EXCSAT SRVNAM parameter
2C	QWHDPRID	The ACCRDB PRDID parameter. This is the product ID of the application requester. Private protocols will set this field to an appropriate product ID value. These ID values are zero if the header is written at the application requester site.
34	QWHDRQNM_Off	If QWHDRQNM is truncated, this is the offset from the beginning of QWHD to QWHDRQNM_LEN. If zero, refer to QWHDRQNM.
Defined by QWHDRQNM_Off	QWHDRQNM_D	This field contains both QWHDRQNM_Len and QWHDRQNM_Var.
Defined by QWHDRQNM_Off	QWHDRQNM_Len	Length of the field
Defined by QWHDRQNM_Off	QWHDRQNM_Var	The requester location name.
36	QWHDSVNM_Off	If QWHDSVNM is truncated, this is the offset from the beginning of QWHD to QWHDSVNM_LEN. If zero, refer to QWHDSVNM
Defined by QWHDSVNM_Off	QWHDSVNM_D	This field contains both QWHDSVNM_Len and QWHDSVNM_Var.
Defined by QWHDSVNM_Off	QWHDSVNM_Len	Length of the field
Defined by QWHDSVNM_Off	QWHDSVNM_Var	The SRVNAM parameter of the DRDA EXCSAT command.
38	QWHDEND	End of distributed header

The following table shows the contents of the trace header.

Table 171. Contents of trace header

Hex Offset	Macro DSNDQWHT field	Description
0	QWHTLEN	Length of the trace header
2	QWHTTYP	Header type
3	QWHTFLG0	Flags.
4	QWHTTID	Event ID

Table 171. Contents of trace header (continued)

Hex Offset	Macro DSNDQWHT field	Description
6	QWHTTAG	ID specified on DSNWTRC macro
7	QWHTFUNC	Reserved.
8	QWHTEB	Execution block address
C	QWHTPASI	Primary address space ID - EPAR
E	QWHTR14A	Register 14 address space ID
10	QWHTR14	Contents of register 14
14	QWHTR15	Contents of register 15
18	QWHTR0	Contents of register 0
1C	QWHTR1	Contents of register 1
20	QWHTEXU	Address of MVS execution unit
24	QWHTDIM	Number of data items
26	QWHTHASI	Home address space ID
28	QWHTFUNCG	The trace function that is set by the DSNWTRC macro.
2C	QWHTDATA	Address of the data
30	QWHTFLAG	Flags in the trace list
32	QWHTDATL	Length of the data list
34	QWHTEND	End of header

The following table shows the contents of the data sharing header.

Table 172. Contents of data sharing header

Hex Offset	Macro DSNDQWHA field	Description
0	QWHALEN	Length of the data sharing header
2	QWHATYP	Header type
3		Reserved
4	QWHAMEMN	DB2 member name
C	QWHADSGN	DB2 data sharing group name
14	QWHAEND	End of header

Figure 67 on page 817 is a sample accounting trace for a distributed transaction sent to SMF.

```
                                                                       A
+0000  093E0000 5E650080 B3FE0108 289FF3F0  F9F0E5F9 F1C20000 00000000 00000818
       B   C    D        E   F    G                  H                 I
+0020  01260001 00000084 02680001 000005A4  01F40001 00000798 00400002 0000054C
                J                 K                  L                 M
+0040  00580001 00000360 00000001 0000044A  01020001 00000000 00000000 00000000
                N
+0060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 000002EC
+0080  00740001 C3269340 28190511 C3269340  EC195F5C 00000000 00888EA0 00000000
+00A0  038D7340 00000000 00000000 00000000  00000000 0000000C 40404040 40404040
+00C0  00000000 00000000 00000001 00000001  00000000 9822C7F8 00000000 02C55F40
+00E0  00000000 00000000 00000000 144C65E3  00000000 00000000 0000001A 0000002A
+0100  00000000 00000000 00000000 00000000  00000000 01190F81 00000000 00000000
+0120  00000000 00000000 00000000 00000002  00000000 00000000 00000000 00000000
+0140  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0160  00000000 003F0001 00000000 00000000  00000000 00000000 00000000 00000000
```

```
+0180  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+01A0  00000000 00000000 000017CF C1D3D3E3  E2D64040 00000000 00000000 00000000
+01C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+01E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0200  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0220  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0240  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0260  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0280  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+02A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+02C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+02E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0300  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0320  1AE7A27D 00000000 00BAFCAB 00000000  0A492E01 00000000 00000000 0000000C
+0340  00000004 0000000C 00000000 00000000  00000000 00000000 00000000 00000000
       O
+0360  00E8E2E3 D3C5C3F1 40404040 40404040  40400000 00000000 00080000 00040000
+0380  00000000 0B950000 06090000 00000000  00010000 000A0000 000A0000 00000000
+03A0  00010000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+03C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00060000
+03E0  00000000 00000000 00000000 00008000  00000000 00000000 00010000 00000000
+0400  00010000 00000000 00000000 00000000  00000000 00010000 00000000 00000000
+0420  00000000 00000000 00000000 00000000  00010000 0001C4E2 D5F0F9F0 F1F00000
                             P
+0440  00000000 00000000 00005FC4 E2D5F0F9  F0F1F5E2 E3D3C5C3 F1404040 40404040
+0460  404040E4 E2C9C2D4 E2E840E2 E8C5C3F1  C4C2F2C2 C1E3C3C8 404040C2 C1E3C3C8
+0480  404040E3 C5D7F440 40404040 404040E2  E8E2C1C4 D44040C4 E2D5E3C5 D7F340E4
+04A0  E2C5D97E E2E8E2C1 C4D44040 40404040  40404040 40404040 40404040 40404040
+04C0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
+04E0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
+0500  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
+0520  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
                                  Q
+0540  40404040 40404040 40400000 00000000  00000000 00000000 00000000 00000000
+0560  00000003 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0580  0000001C 00000016 00000000 00000001  00000000 00000008 00000000 00000000
                R
+05A0  00000000 209501F4 D8E7E2E3 00000000  00000000 00000000 00000000 00000001
+05C0  00000001 00000001 00000000 00000000  00000000 00000000 00000000 00000000
+05E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0600  00000000 00000000 00000005 00000000  00000000 00000000 00000000 00000000
+0620  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0640  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0660  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0680  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+06A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+06C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+06E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0700  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0720  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0740  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
+0760  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
                                                              S
+0780  00000000 00000000 00000000 00000000  00000000 00000000 00000000 0000001A
+07A0  00000000 00000000 0000000F 00000002  00000000 00000000 00000000 00000000
+07C0  00000000 00000000 00000000 00000000  00000001 00000000 00000064 0000000F
+07E0  00000000 00000000 00000007 00000000  00000000 00000000 00000000 00000000
                                                              T
+0800  00000000 00000000 00000000 00000000  00000000 00000000 0052011A 00030D91
+0820  16258770 E5F9F1C2 C3269340 EC55F159  00000001 00000007 00000004 E2E3D3C5
+0840  C3F1C240 40404040 40404040 E4E2C9C2  D4E2E840 E2E8C5C3 F1C4C2F2 C326933F
                             U
+0860  0F3E0003 00000000 0001009C 0200E2E8  E2C1C4D4 4040E3C5 D7F44040 40404040
+0880  4040C2C1 E3C3C840 4040C4E2 D5E3C5D7  F340E2E8 E2C1C4D4 40400000 0008E4E2
+08A0  C9C2D4E2 E84BE2E8 C5C3F1C4 C2F2C326  933F0F3E 00004040 40404040 40404040
+08C0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040
+08E0  40404040 40404040 40404040 40404040  40404040 40404040 00000000 00000000
```

```
                      V
+0900  00000000 00000038 1000E2E3 D3C5C3F1  40404040 40404040 4040C326 9340D6D7
+0920  C790E2E3 D3C5C3F1 40404040 40404040  4040C4E2 D5F0F9F0 F1F00000 0000
```

Figure 67. DB2 distributed data trace output sent to SMF (printed with IMS DFSERA10 print program). This example has one accounting record (IFCID 0003) from the server site (SILICON_VALLEY_LAB). DSNDQWA0 maps the self-defining section for IFCID 0003.

Key to Figure 67	Description
A 00000818	Offset to product section; beginning of self-defining section
B 0126	Length of product section
C 0001	Number of times product section is repeated
D 00000084	Offset to accounting section
E 0268	Length of accounting section
F 0001	Number of times accounting section is repeated
G 000005A4	Offset to SQL accounting section
H 00000798	Offset to buffer manager accounting section
I 0000054C	Offset to locking accounting section
J 00000360	Offset to distributed section
K 0000044A	Offset to MVS/DDF accounting section
L 00000000	Offset to IFI accounting section
M 00000000	Offset to package accounting section
N 00000000	Beginning of accounting section (DSNDQWAC)
O 00E8E2E3	Beginning of distributed section (DSNDQLAC)
P 00005FC4	Beginning of MVS/DDF accounting section (DSNDQMDA)
Q 00000000	Beginning of locking accounting section (DSNDQTXA)
R 209501F4	Beginning of SQL accounting section (DSNDQXST)
S 00000000	Beginning of buffer manager accounting section (DSNDQBAC)
T 0052011A	Beginning of product section (DSNDQWHS); beginning of standard header
U 0001009C	Beginning of correlation header (DSNDQWHC)
V 00000038	Beginning of distributed header (DSNDQWHD)

PSPI

Trace field descriptions

You can find descriptions of trace records in *prefix*.SDSNIVPD(DSNWMSGS).

PSPI

A DB2 trace record consists of a number of trace fields. Each trace record has an associated instrumentation facility component ID (IFCID), and each trace field has a unique name. The DSNWMSGS file contains a list of the trace records, ordered by IFCID, and a description of each field within a trace record. The DSNWMSGS file includes instructions for creating a table into which you can load its contents. After you load the data, you can use SQL queries to search for specific IFCIDs and descriptions of their contents. You can also use SQL queries to retrieve summaries of trace types, classes, and associated IFCIDs.

If you intend to write a program to read DB2 trace records, use the assembler mapping macros in *prefix*.SDSNMACS. The macros have member names that begin with DSNDQW.

You can use the TSO or ISPF browse function to look at the field descriptions in the trace record mapping macros online, even when DB2 is down. If you prefer to look at the descriptions in printed form, you can use ISPF to print a listing of the data set.

PSPI

Related concepts:

Types of DB2 traces

Related tasks:

Controlling traces (DB2 Administration Guide)

Related reference:

Instrumentation facility interface (IFI) records

Chapter 47. Programming for the instrumentation facility interface (IFI)

Monitor programs can use the *instrumentation facility interface (IFI)* to request online trace information from the trace facility. IFI can be accessed through any of the attachment facilities. It gathers trace data that can be written to one or more destinations that you specify.

PSPI

You can use the trace information for the following purposes:

- To obtain accounting information for online billing.
- To periodically obtain system-wide information about DB2, highlight exceptional conditions, or provide throughput information.
- To learn which processes have been connected to DB2 the longest, or which processes have used the most CPU time.
- To obtain accounting records as transactions terminate.
- To determine the access and processing methods for an SQL statement.
- To capture log buffers online for use in remote recovery.
- To retrieve SQL changes synchronously from the log, for processing in an application.

IFI uses the standard security mechanisms such as connection authorization, plan authorization, and so forth.

Before using IFI, you should be familiar with the trace facility and instrumentation facility component identifiers (IFCIDs).

Note that where the trace output indicates a particular release level, you see '*xx*' to show that this information varies according to the actual release of DB2 that you are using.

You can use IFI in a monitor program (a program or function outside of DB2 that receives information about DB2).

When a trace is active, internal events trigger the creation of trace records. The records, identified by *instrumentation facility component identifiers* (IFCIDs), can be written to buffers, and a monitor program can read them later by using the IFI READA function. READA requests are *asynchronous*, because the data is not read by the monitor program at the same time that it was written.

You can trigger the creation of certain types of trace records by using the IFI READS function. READS requests are *synchronous*, which means that the records are not held in a buffer. Instead, the records are returned directly to the return area for the monitor program.. The data is collected at the time of the request for the data.

PSPI

Related concepts:

DB2 trace

Related tasks:

Planning for and designing DB2 applications (DB2 Application programming and SQL)

Overview of programming applications that access DB2 for z/OS data (DB2 Application programming and SQL)

Preparing an application to run on DB2 for z/OS (DB2 Application programming and SQL)

Related reference:

Instrumentation facility interface (IFI) records

IFCID Record Blocks (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Invoking IFI from a monitor program

You can issue calls to IFI functions from a program or function outside of DB2 to submit commands, obtain trace data, and pass data to DB2 through IFI.

Procedure

PSPI

To invoke IFI from a program:

1. Include a call to DSNWLI in your monitor program. You can invoke IFI from assembler and PL/I programs. The following example depicts an IFI call in an assembler program. All IFI-related examples are given for assembler.

   ```
   CALL DSNWLI,(function,ifca,parm-1,...parm-n),VL
   ```

   The parameters that are passed on the call indicate the function, point to communication areas used by the function, and provide other information that depends on the function specified. Because the parameter list might vary in length, the high-order bit of the last parameter must be on to signal that it is the last parameter in the list. For example, VL option signals a variable length parameter list and turns on the bit in assembler.

2. Link-edit the program with the correct language interface.

   The DSNULI language interface module can be linked with 31-bit RRS, CAF, TSO, and CICS applications. DSNULI cannot be linked with 24-bit application programs. Each of the following language interface modules also has an entry point of DSNWLI for IFI:

   - CAF DSNALI
   - TSO DSNELI
   - CICS DSNCLI
   - IMS DFSLI000
   - RRSAF DSNRLI

   CAF DSNALI, the CAF (call attachment facility) language interface module, includes a second entry point of DSNWLI2. The monitor program that link-edits DSNALI with the program can make IFI calls directly to DSNWLI. The monitor program that loads DSNALI must also load DSNWLI2 and remember its address. When the monitor program calls DSNWLI, the program must have a dummy entry point to handle the call to DSNWLI and then call the real DSNWLI2 routine.

What to do next

A monitor program has the following requirements:

Connection
A monitor program that issues IFI requests must be connected to DB2 at the thread level. If the program contains SQL statements, you must precompile the program and create a DB2 plan using the BIND process. If the monitor program does not contain any SQL statements, it does not have to be precompiled. However, as is the case in all the attachment environments, even though an IFI only program (one with no SQL statements) does not have a plan of its own, it can use any plan to get the thread level connection to DB2

Bit mode
The monitor program can run in either 24- or 31-bit mode.

Authorization
On the first READA or READS call from a user, an authorization is checked to determine if the primary authorization ID or one of the secondary authorization IDs of the plan executor has MONITOR1 or MONITOR2 privilege. If your installation uses the access control authorization exit routine, that exit routine might control the privileges that can use the monitor trace. If you have an authorization failure, an audit trace (class 1) record is generated that contains the return and reason codes from the exit. This is included in IFCID 0140.

PSPI

.

Related concepts:

Monitor trace

Accounting trace

Common communication areas for IFI calls

Access control authorization exit routine (Managing Security)

Related tasks:

Invoking the call attachment facility (DB2 Application programming and SQL)

Related reference:

IFI functions

Submitting commands from monitor programs

Monitor programs can call the COMMAND function to issue DB2 commands through the instrumentation facility interface. This capability is most useful for submitting commands to start, stop, display, and modify traces.

PSPI

Using specified trace classes and IFCIDs, a monitor program can control the amount and type of its data. You can design your monitor program to:

- Activate and deactivate pre-defined trace classes.

- Activate and deactivate a trace record or group of records (identified by IFCIDs).
- Activate and deactivate predefined trace classes and trace records (identified by IFCIDs) restricting tracing to a set of identifiers, such as plan name, authorization ID, resource manager identifier (RMID), and so on.

The command is processed and the output messages are returned to the monitor program in the return area.

Asynchronous commands through IFI

You can execute commands asynchronously or synchronously. You can set a bit in the IFCA to specify how you want IFI to execute the commands. The following commands are affected:

- ALTER BUFFERPOOL
- SET LOG
- STOP DATABASE
- SET SYSPARM

Example DB2 command through IFI

PSPI

This example issues a DB2 START TRACE command for MONITOR class 1.

Figure 68. Starting a trace using IFI

```
CALL DSNWLI,('COMMAND ',IFCAAREA,RETAREA,OUTAREA,BUFAREA),VL
⋮
COMMAND DC CL8 'COMMAND '
************************************************************************
* Function parameter declaration                                       *
************************************************************************
* Storage of LENGTH(IFCA) and properly initialized                     *
************************************************************************
IFCAAREA  DS       0CL180
⋮
************************************************************************
* Storage for length and returned info.                                *
************************************************************************
RETAREA   DS      CL608
************************************************************************
* Storage for length and DB2 Command                                   *
************************************************************************
OUTAREA   DS      0CL42
OUTLEN    DC      X'002A0000'
OUTCMD    DC      CL38'-STA TRAC(MON) DEST(OPX) BUFSIZE(1024)'
************************************************************************
* Storage of LENGTH(WBUF) and properly initialized                     *
************************************************************************
BUFAREA  DS       0CL16
⋮
```

PSPI

Related reference:

COMMAND

Command record format

-START TRACE (DB2) (DB2 Commands)

-STOP TRACE (DB2) (DB2 Commands)

-DISPLAY TRACE (DB2) (DB2 Commands)

-MODIFY TRACE (DB2) (DB2 Commands)

Writing to trace destinations from monitor programs

A monitor program can write information to a specific trace destination by issuing a WRITE request for a specific IFCID.

About this task

PSPI

A WRITE request is written to a destination that is activated by a START TRACE command.

For example, you can write data to trace destinations for the following purposes:

- To extend accounting data. For example, a monitor program can collect batch file I/O counts, store them in a user-defined trace record, and process them along with standard DB2 accounting data.
- Include accounting data from QMF, IMS, or CICS
- Permit CICS users to write the CICS accounting token and task number into the DB2 trace, assuming ACCOUNTREC in the DB2ENTRY RDO definition is neither UOW nor TASK .

Procedure

When writing information to trace destinations with the WRITE function, use the following approaches:

- Establish usage procedures and standards for the WRITE function. Procedures ensure that the correct IFCIDs are active when the WRITE function is called. Standards determine the records and record formats that a monitor program sends to DB2.
- Place the record type and sub-type in the first fields in the data record. You can use one IFCID to contain many different records, and this information can help you keep track of the data.

PSPI

Related reference:

WRITE

-START TRACE (DB2) (DB2 Commands)

Requesting data asynchronously from a monitor program

You can create a monitor program that requests an asynchronous buffer, which records trace data as trace events occur. Monitor programs can call the READA function to read data asynchronously from an OP*n* buffer.

About this task

PSPI

DB2 can buffer all IFCID data that is activated by the START TRACE command and pass it to a monitor program on a READA request. The IFCID events include the following types of information:

- Serviceability
- Statistics
- Accounting
- Performance
- Audit data
- IFCIDs defined for the IFI write function

Procedure

To create a monitor program that requests asynchronous data, use the following approaches:

- Issue a START TRACE command from the monitor program and specify a buffer location to start the collection of asynchronous data. You can specify a generic value or specific online performance monitor destinations, OP*n*, where *n* is any integer from 1 to 8. After the trace is started, DB2 collects and buffers the information as it occurs.
- Specify the generic OPX option in the initial START TRACE command to avoid conflicts with other traces or programs. When OPX is specified, the instrumentation facility assigns the next available buffer destination slot and returns the OP*n* destination name to the monitor program. The program can use subsequent START TRACE or MODIFY TRACE commands to direct the data to the destination specified by the instrumentation facility.
- Issue READA requests to move the buffered data to the monitor program.
- Use specific initial OP*n* destinations in the following situations:
  - When you plan to start numerous asynchronous traces to the same OP*n* destination. To do this, you must specify the OP*n* destination in your monitor program. The OP*n* destination that is started is returned in the IFCA.
  - When the monitor program specifies a particular monitor class (defined as available) together with a particular destination (for example OP7) to indicate that certain IFCIDs are started. An operator can use the DISPLAY TRACE command to determine which monitors are active and what events are being traced.
- Configure the monitor program to use large buffers and specify small WBUFBC values to prevent data loss. Data loss occurs when the buffer fills before the monitor program can obtain the data. DB2 does not wait for the buffer to be emptied. Instead, it informs the monitor program on the next READA request (in the IFCARLC field of the IFCA) that the data was lost.
- Display the asynchronous data on a terminal by creating a program that uses the following logic:
  1. Initialize.
  2. Use GETMAIN to obtain a storage area equal to the BUFSIZE value in the START TRACE command.
  3. Issue the following command through IFI to wake up this routine by a POST whenever the buffer is 20% full:

     ```
     START TRACE=ACCTG DEST=OPX
     ```

4. Check the status in the IFCA to determine if the command request was successful.
5. WAIT for the buffer to be posted.
6. Clear the post flag.
7. Issue a READA request to obtain the buffer data.
8. Check the status of the READA request.
9. De-block in the information.
10. Display the information on the terminal.
11. Loop back to the WAIT.

PSPI

Related reference:

READA

COMMAND

Instrumentation facility communications area (IFCA)

-START TRACE (DB2) (DB2 Commands)

-DISPLAY TRACE (DB2) (DB2 Commands)

Requesting data synchronously from a monitor program

READS allows your monitor program to read DB2 status information that is collected at the time of the IFI call.

About this task

For a list of trace fields that can be returned by READS requests, see "Trace fields for READS requests" on page 839.

Because of performance considerations, the majority of data that is obtained by a monitor program probably comes over the synchronous interface. Summarized information is easier for a monitor program to process, and the monitor program logic is simpler because a smaller number of records are processed.

Procedure

For monitor programs that collect synchronous data, use the following approaches:

- Consider reasonability tests for data that is obtained through READS. The READS request can reference data that is updated during the retrieval process. Because the READS function does not usually suspend activity that takes place under referenced structures, an abend can occur. If an abend occurs, the READS function is terminated without a dump and the monitor program is notified through the return code and reason code information in the IFCA. However, the return area can contain valid trace records, even if an abend occurred; therefore, your monitor program should check for a non-zero value in the IFCABM (bytes moved) field of the IFCA.
- When you use a READS request with a query parallelism task, remember that each parallel task is a separate thread. Each parallel thread has a separate READS output.A READS request might return thread information for parallel tasks on a DB2 data sharing member without the thread information for the originating task in a Sysplex query parallelism case.

- Use the qual-area parameter, mapped by DSNDWQAL, to qualify the trace records to be returned on IFI READS requests.
- Display the synchronous trace data on a terminal by creating a monitor program that uses the following logic:
  1. Initialize.
  2. Set a timer.
  3. Wait for the timer to expire.
  4. Call IFI to obtain statistics data via a READS request.
  5. Do delta calculations to determine activity. This step is not necessary for IFCID 0199 because the statistics are reset statistics at the beginning of every collection interval.
  6. Display the information on a terminal.
  7. Loop back to the timer.

PSPI

Related reference:

READS

Instrumentation facility interface (IFI) records

Related information:

Reading log records (DB2 Administration Guide)

Monitoring static SQL statements with READS calls

You can use READS requests from an IFI application to monitor static SQL statements.

Procedure

To create an IFI program that monitors static SQL statements uses the following steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start performance trace class monitor class 29 for IFCID 0400. This step enables statistics collection for static SQL statements.
3. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute static SQL statements.
4. Resume the IFI program after enough time has elapsed for a reasonable amount of static SQL statement activity to occur.
5. Set up the qualification area for a READS call for IFCID 0401.
6. Set up the IFCID area to request data for IFCID 0401.
7. Examine the contents of the return area.
8. For a statement with unexpected statistics values:
   a. Obtain the statement ID from the IFCID 0401 return area.
   b. Query the STMT_ID column of the SYSIBM.SYSPACKSTMT catalog table, by using the statement ID, and obtain the statement text from the STATEMENT column.
   c. Use the statement text to execute an SQL EXPLAIN statement.
   d. Fetch the EXPLAIN results from the PLAN_TABLE.
9. Issue an IFI COMMAND call to stop performance trace class 29 for IFCID 0400.

Related concepts:

Investigating SQL performance by using EXPLAIN

Related tasks:

Controlling the collection of statistics for SQL statements

Monitoring the dynamic statement cache with READS calls

Collecting statement-level statistics for SQL statements

Related reference:

READS

Monitoring the dynamic statement cache with READS calls

You can use READS requests from an IFI application to monitor the contents of the dynamic statement cache, and optionally, to see some accumulated statistics for those statements.

About this task

PSPI

This strategy can help you detect and diagnose performance problems for those cached dynamic SQL statements.

An IFI program that monitors the dynamic statement cache should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start monitor class 29. This step enables statistics collection for statements in the dynamic statement cache.
3. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute dynamic SQL statements by using the dynamic statement cache.
4. Resume the IFI program after enough time has elapsed for a reasonable amount of activity to occur in the dynamic statement cache.
5. Set up the qualification area for a READS call for IFCID 0316
6. Set up the IFCID area to request data for IFCID 0316.
7. Issue an IFI READS call to retrieve the qualifying cached SQL statements.
8. Examine the contents of the return area.

   For a statement with unexpected statistics values:

   a. Obtain the statement name and statement ID from the IFCID 0316 data.
   b. Set up the qualification area for a READS call for IFCID 0317.
   c. Set up the IFCID area to request data for IFCID 0317.
   d. Issue a READS call for IFCID 0317 to get the entire text of the statement.
   e. Obtain the statement text from the return area.
   f. Use the statement text to execute an SQL EXPLAIN statement.
   g. Fetch the EXPLAIN results from the PLAN_TABLE.
9. Issue an IFI COMMAND call to stop performance trace class 29 for IFCID 0318.

PSPI

Related concepts:

Investigating SQL performance by using EXPLAIN

Related tasks:

Capturing performance information for dynamic SQL statements

Controlling the collection of statistics for SQL statements

Collecting statement-level statistics for SQL statements

Monitoring static SQL statements with READS calls

Related reference:

READS

Monitoring deadlocks and timeouts from a monitor program

Monitor programs can call IFI functions to monitor deadlocks and timeouts.

About this task

An IFI program that monitors deadlocks and timeouts of cached statements might include the following steps:

PSPI

Procedure

To monitor deadlocks and timeouts through the instrumentation facility interface:

Create a monitor program that uses the following actions:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Start statistics trace class 3, or performance trace class 6, for IFCID 0172 to monitor deadlocks, or for IFCID 0196 to monitor timeouts.
3. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute dynamic SQL statements by using the dynamic statement cache.
4. Resume the IFI program when a deadlock or timeout occurs.
5. Issue a READA request to obtain IFCID 0172 or IFCID 0196 trace data.
6. Obtain the cached statement ID of the statement that was involved in the deadlock or timeout from the IFCID 0172 or IFCID 0196 trace data. Using the statement ID, set up the qualification area for a READS call for IFCID 0316 or IFCID 0317..
7. Set up the IFCID area to request data for IFCID 0316 or IFCID 0317.
8. Issue an IFI READS call to retrieve the qualifying cached SQL statement.
9. Examine the contents of the return area.
10. Issue an IFI COMMAND call to stop statistics trace class 3 for IFCID 0172 or IFCID 0196.

Related concepts:

Lock contention

Related tasks:

Controlling the collection of statistics for SQL statements

Monitoring concurrency and locks

Related reference:

IFI functions

Controlling the collection of statistics for SQL statements

The collection of statement-level statistics for SQL statements might increase the processing costs for those statements.

Before you begin

Monitor trace class 29 must be active to enable the collection of statement-level statistics.

Procedure

To minimize the costs of collection statistics for SQL statements:

- Use IFCID 0318 to enable and disable the collection of dynamic statement cache statistics in IFCID 0316.

  When IFCID 0318 is inactive, DB2 does not collect those statistics. DB2 tracks the statements in the dynamic statement cache, but does not accumulate the statistics as those statements are used. When you stop or start the trace for IFCID 0318, DB2 resets the IFCID 0316 statistics counters for all statements in the cache to 0.

  If you issue a READS call for IFCID 0316 while IFCID 0318 is inactive, DB2 returns identifying information for all statements in the cache, but returns 0 in all the IFCID 0316 statistics counters. When you are not actively monitoring the cache, you should turn off the trace for IFCID 0318.
- Use IFCID 0400 to enable and disable the collection of statistics for static SQL statements through IFCID 0401.

  When IFCID 0400 is inactive, DB2 does not collect those statistics. DB2 tracks the statements in the EDM pool, but does not accumulate the statistics as those statements are used. When you stop or start the trace for IFCID 0400, DB2 resets the IFCID 0401 statistics counters for all static SQL statements to 0.

  If you issue a READS call for IFCID 0401 while IFCID 0400 is inactive, DB2 returns identifying information for all statements in the EDM pool, but returns 0 in all the IFCID 0316 statistics counters. When you are not actively monitoring statistics SQL statements, you should turn off the trace for IFCID 0400.

Related tasks:

Monitoring the dynamic statement cache with READS calls

Collecting statement-level statistics for SQL statements

Monitoring static SQL statements with READS calls

Monitoring SQL performance with IBM optimization tools

Using IFI from stored procedures

You can use the IFI interface from a stored procedure, and the output of the trace can be returned to the client.

About this task

PSPI

You can also issue DB2 commands, such as “DISPLAY THREAD”, from a stored procedure and return the results to the client.

PSPI

Using IFI in a data sharing group

You can use IFI READA and READS calls in an application that runs on one member of a data sharing group to gather trace data from other members of the data sharing group.

PSPI

You can also use an IFI COMMAND call to execute a command at another member of a data sharing group. In addition to the IFCA fields that you use for an IFI request for a single subsystem, you need to set or read the following fields for an IFI request for a data sharing group:

IFCAGLBL

Set this flag on to indicate that the READS or READA request should be sent to all members of the data sharing group.

IFCADMBR

If you want an IFI READS, READA, or COMMAND request to be executed at a single member of the data sharing group, assign the name of the group member to this field. If you specify a name in this field, DB2 ignores IFCAGLBL. If the name that you specify is not active when DB2 executes the IFI request, DB2 returns an error.

Recommendation: To issue a DB2 command that does not support SCOPE(GROUP) at another member of a data sharing group, set the IFCADMBR field and issue an IFI COMMAND.

IFCARMBR

The name of the data sharing member that generated the data that follows the IFCA. DB2 sets this value in the copy of the IFCA that it places in the requesting program's return area.

IFCAGRSN

A reason code that DB2 sets when not all data is returned from other data sharing group members.

IFCAGBM

The number of bytes of data that other members of the data sharing group return and that the requesting program's return area can contain.

IFCAGBNM

The number of bytes of data that members of the data sharing group return but that the requesting program's return area cannot contain.

As with READA or READS requests for single DB2 subsystems, you need to issue a START TRACE command before you issue the READA or READS request. You can issue START TRACE with the parameter SCOPE(GROUP) to start the trace at all members of the data sharing group. For READA requests, specify DEST(OPX) in the START TRACE command. DB2 collects data from all data sharing members and returns it to the OPX buffer for the member from which you issue the READA request.

If a new member joins a data sharing group while a trace with SCOPE(GROUP) is active, the trace starts at the new member.

After you issue a READS or READA call for all members of a data sharing group, DB2 returns data from all members in the requesting program's return area. Data from the local member is first, followed by the IFCA and data for all other members.

Example

If the local DB2 is called DB2A, and the other two members in the group are DB2B and DB2C, the return area looks like this:

```
Data for DB2A
IFCA for DB2B (DB2 sets IFCARMBR to DB2B)
Data for DB2B
IFCA for DB2C (DB2 sets IFCARMBR to DB2C)
Data for DB2C
```

If an IFI application requests data from a single other member of a data sharing group (IFCADMBR contains a member name), the requesting program's return area contains the data for that member but no IFCA for the member. All information about the request is in the requesting program's IFCA.

Because a READA or READS request for a data sharing group can generate much more data than a READA or READS request for a single DB2, you need to increase the size of your return area to accommodate the additional data.

PSPI

Related concepts:

X'E6......' codes (DB2 Codes)

Related tasks:

Requesting data asynchronously from a monitor program

Requesting data synchronously from a monitor program

Related reference:

-START TRACE (DB2) (DB2 Commands)

Instrumentation facility communications area (IFCA)

COMMAND

READA

READS

Data integrity and IFI

Although IFI displays DB2 statistics, agent status, and resource status data, it does not change or display DB2 database data.

PSPI

When a process retrieves data, information is moved from DB2 fetch-protected storage to the user's address space, or from the address space to DB2 storage, in the storage key of the requester. Data that is moved by the READA request is serialized so that only “clean data” is moved to the address space of the requester.

The serialization techniques used to obtain data for a given READA request might minimally degrade performance on processes that simultaneously store data into the instrumentation facility buffer. Failures during the serialization process are handled by DB2.

The DB2 structures that are searched on a READS request are validated before they are used. If the DB2 structures are updated while being searched, inconsistent data might be returned. If the structures are deleted while being searched, users might access invalid storage areas, causing an abend. If an abend does occur, the functional recovery routine of the instrumentation facility traps the abend and returns information about it to the application program's IFCA.

PSPI

Related tasks:

Protecting data integrity (Managing Security)

Requesting data asynchronously from a monitor program

Requesting data synchronously from a monitor program

Related reference:

Instrumentation facility communications area (IFCA)

READA

READS

Auditing data and IFI

Starting, stopping, or modifying trace through IFI might cause changes to the events being traced for audit.

PSPI

Each time these trace commands are processed a record is sent to the destination processing the trace type. In the case of audit, the audit destination receives a record indicating a trace status has been changed. IFCID 0004 and 0005 contain these records.

PSPI

Related concepts:

Auditing access to DB2 (Managing Security)

Submitting commands from monitor programs

Related reference:

-START TRACE (DB2) (DB2 Commands)

-STOP TRACE (DB2) (DB2 Commands)

-MODIFY TRACE (DB2) (DB2 Commands)

Improving concurrency for IFI

When you design your application to use IFI, consider the potential for locking delays, including suspensions, timeouts and deadlocks.

About this task

PSPI

Locks are obtained for IFI in the following situations:

- When READS and READA requests are checked for authorization, short duration locks on the DB2 catalog are obtained. When the check is made, subsequent READS or READA requests are not checked for authorization. Remember, if you are using the access control exit routine, then that routine might be controlling the privileges that the monitor trace can use.
- When DB2 commands are submitted, each command is checked for authorization. DB2 database commands obtain additional locks on DB2 objects.

A program can issue SQL statements through an attachment facility and DB2 commands through IFI. This environment creates the potential for an application to deadlock or timeout with itself over DB2 locks acquired during the execution of SQL statements and DB2 database commands.

Procedure

To ensure that all DB2 locks acquired by preceding SQL statements are no longer held when the DB2 database command is issued:

- Bind the application with the RELEASE(COMMIT) bind option.
- Initiate a commit or rollback to free any locks your application holds, before issuing the command.
- Ensure that time between commit operations is short if your application uses SQL.

PSPI

Related concepts:

Lock contention

Related tasks:

Improving concurrency

Configuring subsystems for concurrency

Designing databases for concurrency

Programming for concurrency

Choosing a RELEASE option

Recovery considerations for IFI

When an application program issues an IFI call, the requested function is immediately performed. However, if the application program subsequently abends, the IFI request is not backed out.

PSPI

In contrast, requests that do not use IFI are committed and abended as usual. For example, if an IFI application program also issues SQL calls, a program abend causes the SQL activity to be backed out.

PSPI

Errors and IFI

You might encounter certain errors while using the instrumentation facility interface (IFI).

PSPI

- Connection failure, because the user is not authorized to connect to DB2
- Authorization failure, because the process is not authorized to access the DB2 resources specified

IFI request failures

Requests sent through IFI can fail for a variety of reasons, including:

- One or more parameters are invalid.
- The IFCA area is invalid.
- The specified OP*n* is in error.
- The requested information is not available.
- The return area is too small.

Return code and reason code information is stored in the following instrumentation facility communication area (IFCA) fields:

IFCARC1
Return codes.

IFCARC2
Reason codes.

PSPI

Related concepts:

X'E6......' codes (DB2 Codes)

Related reference:

Instrumentation facility communications area (IFCA)

IFI functions

Monitor programs can use IFI functions to issue commands, to request trace field data, and to write data to trace fields through the instrumentation facility interface (IFI).

COMMAND

You can use COMMAND functions in monitor programs to issue commands from the instrumentation facility interface (IFI).

PSPI

You can submit any DB2 command, including the following trace commands:

- START TRACE
- STOP TRACE
- DISPLAY TRACE
- MODIFY TRACE

You can execute commands asynchronously or synchronously. You can set a bit in the IFCA to specify how you want IFI to execute the commands. The following commands are affected:

- ALTER BUFFERPOOL
- SET LOG
- STOP DATABASE
- SET SYSPARM

Authorization

The primary authorization ID or one of the secondary authorization IDs of the process must have the appropriate authorization to issue the command. Otherwise the request is denied. An application program might have the authorization to issue DB2 commands, but not the authorization to issue READA requests.

In a data sharing group, if a command is issued through the IFI interface and executes on a different data sharing member from the member the issued the command, then the command executes under the user ID of the application process that issued the IFI call.

Syntax and options

A call to the COMMAND function has the following syntax:

```
CALL DSNWLI,('COMMAND ',ifca,return-area,output-area,buffer-info .),VL
```

ifca

IFCA (instrumentation facility communication area) is an area of storage that contains the return code and reason code. IFCA indicates the following information:

- The success or failure of the request
- Diagnostic information from the DB2 component that executed the command
- The number of bytes moved to the return area
- The number of bytes of the message segments that did not fit in the return area

Some commands might return valid information despite a non-zero return code or reason code. For example, the DISPLAY DATABASE command might indicate that more information could have been returned than was allowed.

If multiple errors occur, the last error is returned to the caller. For example, if the command is in error and the error message does not fit in the area, the error return code and reason code indicate that the return area is too small.

If a monitor program issues START TRACE, the ownership token (IFCAOWNR) in the IFCA determines the owner of the asynchronous buffer. The owner of the buffer is the only process that can obtain data through a subsequent READA request.

return-area

When the issued command finishes processing, it places messages (if any) in the return area. The messages are stored as varying-length records, and the total number of bytes in the records is placed in the IFCABM (bytes moved) field of the IFCA. If the return area is too small, as many message records as can fit are placed into the return area.

The monitor program should analyze messages that are returned by the command function.

output-area
Contains the varying-length command.

buffer-info
This parameter is required for starting traces to an OP buffer. Otherwise, it is not needed. This parameter is used only on COMMAND requests. It points to an area that contains information about processing options when a trace is started by an IFI call to an unassigned OP*n* destination buffer. An OP*n* destination buffer is considered unassigned if it is not owned by a monitor program.

If the OP*n* destination buffer is assigned, then the buffer information area is not used on a later START or MODIFY TRACE command to that OP*n* destination. .

When you use *buffer-info* on START TRACE, you can specify the number of bytes that can be buffered before the monitor program ECB is posted. The ECB is posted when the amount of trace data collected has reached the value that is specified in the byte count field. The byte count field is also specified in the buffer information area.

The following table summarizes the fields in the buffer information area.

Table 173. Buffer information area fields. This area is mapped by assembler mapping macro DSNDWBUF.

Name	Hex offset	Data type	Description
WBUFLEN	0	Signed two-byte integer	Length of the buffer information area, plus 4. A zero indicates the area does not exist.
	2	Signed two-byte integer	Reserved.
WBUFEYE	4	Character, 4 bytes	Eye catcher for block, WBUF.
WBUFECB	8	Address	The ECB address to post when the buffer has reached the byte count specification (WBUFBC). The ECB must reside in monitor key storage. A zero indicates not to post the monitor program. In this case, the monitor program should use its own timer to determine when to issue a READA request.
WBUFBC	C	Signed four-byte integer	The records placed into the instrumentation facility must reach this value before the ECB can be posted. If the number is zero, and an ECB exists, posting occurs when the buffer is full.

Example COMMAND function call

This example issues a DB2 START TRACE command for MONITOR Class 1.

```
CALL DSNWLI,('COMMAND ',IFCAAREA,RETAREA,OUTAREA,BUFAREA),VL
⋮
COMMAND DC CL8 'COMMAND '
************************************************************************
* Function parameter declaration                                       *
************************************************************************
* Storage of LENGTH(IFCA) and properly initialized                     *
************************************************************************
IFCAAREA  DS       0CL180
⋮
************************************************************************
* Storage for length and returned info.                                *
************************************************************************
RETAREA   DS      CL608
************************************************************************
* Storage for length and DB2 Command                                   *
************************************************************************
OUTAREA   DS      0CL42
OUTLEN    DC      X'002A0000'
OUTCMD    DC      CL38'-STA TRAC(MON) DEST(OPX) BUFSIZE(1024)'
************************************************************************
* Storage of LENGTH(WBUF) and properly initialized                     *
************************************************************************
BUFAREA  DS       0CL16
⋮
```

Figure 69. Starting a trace using IFI

PSPI

Related concepts:

Submitting commands from monitor programs

Related tasks:

Issuing commands from application programs (DB2 Administration Guide)

Granting authorization on DB2 commands (Managing Security)

Related reference:

Instrumentation facility communications area (IFCA)

System privileges (RACF Access Control Module Guide)

Related information:

About DB2 and related commands (DB2 Commands)

READA

Monitor programs can issue READA requests to request asynchronous data from the instrumentation facility interface.

Authorization

PSPI

On a READA request, the application program must own the specified destination buffer, or the request is denied. You can obtain ownership of a storage buffer by issuing a START TRACE to an OP*n* destination. If the primary authorization ID or one of the secondary authorization IDs of the process does not have MONITOR1 or MONITOR2 privilege, the request is denied. READA requests are checked for authorization once for each user of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

Syntax and options

READA calls to the instrumentation facility interface have the following syntax:

```
CALL DSNWLI,('READA   ',ifca,return-area),VL
```

ifca

Contains information about the OP*n* destination and the ownership token value (IFCAOWNR) at call initiation. After the READA call completes, the IFCA contains the return code, reason code, the number of bytes moved to the return area, the number of bytes not moved to the return area if the area was too small, and the number of records lost.

return-area

Contains the varying-length records that are returned by the instrumentation facility. If the return area is too small, as much of the output as can fit is placed into the area (a complete varying-length record). Reason code 00E60802 is returned in cases where the monitor program's return area is not large enough to hold the returned data.

IFI allocates up to eight OP buffers upon request from private storage in the DB2 MSTR address space. IFI uses these buffers to store trace data until the owning application performs a READA request to transfer the data from the OP buffer to the application's return area. An application becomes the owner of an OP buffer when it issues a START TRACE command and specifies a destination of OPn or OPX. Each buffer can be of size 256 KB to 16 MB. IFI allocates a maximum of 16 MB of storage for each of the eight OP buffers. The default monitor buffer size is determined by the MONSIZE subsystem parameter.

PSPI

Related tasks:

Requesting data asynchronously from a monitor program

Related reference:

Instrumentation facility communications area (IFCA)

Return area

System privileges (RACF Access Control Module Guide)

MONITOR SIZE field (MONSIZE subsystem parameter) (DB2 Installation and Migration)

-START TRACE (DB2) (DB2 Commands)

Related information:

00E60802 (DB2 Codes)

READS

Monitor programs can call the READS function to request synchronous data from the instrumentation facility interface.

Authorization

PSPI

The primary authorization ID or one of the secondary authorization IDs of the process running the application program must have MONITOR1 or MONITOR2 privilege. If neither the primary authorization ID nor one of the secondary authorization IDs has authorization, the request is denied.

READS requests are checked for authorization once for each user (ownership token) of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

If you use READS to obtain your own data (IFCID 0124, 0147, 0148, or 0150 not qualified), no authorization check is performed.

Syntax and options

READS calls to the instrumentation facility interface have the following syntax:

```
CALL DSNWLI,('READS   ',ifca,return-area,ifcid-area,qual-area),VL
```

ifca
: Contains information about the status of the READS call.

return-area
: Contains the varying-length records that are returned by the instrumentation facility. If the specified return area is too small to hold all of the records that are returned, it contains as many records as can fit. The monitor program obtains the return area for READS requests in its private address space.

ifcid-area
: Contains the IFCIDs of the information that you want. The number of IFCIDs can be variable. If the length specification of the IFCID area is exceeded or an IFCID of X'FFFF' is encountered, the list is terminated. If an invalid IFCID is specified, no data is retrieved.

qual-area
: The address of the qualification area where the monitor program can specify constraints on the data that is to be returned. This parameter is optional, and is used only on READS requests.

Related tasks:

Requesting data synchronously from a monitor program

Related reference:

Instrumentation facility communications area (IFCA)

IFCID area

Qualification fields for READS requests

⇨ System privileges (RACF Access Control Module Guide)

Trace fields for READS requests

A monitor program can call the READS function to request synchronous data from certain specific trace records.

PSPI

The following trace records can be requested in READS calls to the instrumentation facility interface:

0001 Statistical data on the systems services address space, including

- Task control block (TCB) and service request block (SRB) times for system services
- Database services, including DDF statistics
- Internal Resource Lock Manager (IRLM) address spaces
- Usage information by LPAR, DB2 subsystem, or DB2 address space
- CPU utilization
- Storage metrics

0002 Statistical data on the database services address space.

0104 Information on the current active log data sets.

0106 Static system parameters.

0124[1] An active SQL snapshot that provides status information about:

- The process
- The SQL statement text
- The relational data system input parameter list (RDI) block
- Certain bind and locking information

You can obtain a varying amount of data because the request requires the process to be connected to DB2, have a cursor table allocated (RDI and status information is provided), and be active in DB2 (SQL text is provided if available). The SQL text that is provided does not include the SQL host variables, and is truncated at 4000 bytes.

For dynamic SQL, IFI provides the original SQL statement. The RDISTYPE field contains the actual SQL function taking place. For example, for a SELECT statement, the RDISTYPE field can indicate that an open cursor, fetch, or other function occurred. For static SQL, you can see the DECLARE CURSOR statement, and the RDISTYPE indicates the function. The RDISTYPE field is mapped by mapping macro DSNXRDI.

0129[1] Returns one or more VSAM control intervals (CIs) that contain DB2 recovery log records. You can use IFI to return these records for use in remote site recovery.

0147[1] An active thread snapshot that provides a status summary of processes at a DB2 thread or non-thread level.

0148[1] An active thread snapshot that provides more detailed status of processes at a DB2 thread or non-thread level.

0149[1] Information that indicates who (the thread identification token) is holding locks and waiting for locks on a particular resource and hash token. The data is in the same format as IFCID 0150.

0150[1] All the locks held and waited on by a given user or owner (thread identification token).

0185 Data descriptions for each table for which captured data is returned on this DATA request. IFCID 0185 data is only available through a propagation exit routine that is triggered by DB2.

0199[1] Information about buffer pool usage by DB2 data sets. DB2 reports this information for an interval that you specify in the DATASET STATS TIME field of installation panel DSNTIPN. At the beginning of each interval, DB2 resets these statistics to 0.

0202 Dynamic system parameters.

0217 Storage detail record for the DBM1 address space.

0225 Storage summary record for the DBM1 address space.

0230 Global statistics for data sharing.

0234[1] User authorization information.

0254[1] Group buffer pool usage in the data sharing group.

0306 Returns compressed or decompressed log records in both a data sharing or non data-sharing environment. For more information about using IFCID 0306 to read log records, see Reading complete log data (IFCID 0306) (DB2 Administration Guide).

0316[1] Returns information about the contents of the dynamic statement cache. The IFI application can request information for all statements in the cache, or provide qualification parameters to limit the data returned. DB2 reports the following information about a cached statement:

- A statement name and ID that uniquely identify the statement
- If IFCID 0318 is active, performance statistics for the statement
- The first 60 bytes of the statement text

0317 Returns the complete text of an SQL statement in the dynamic statement cache and the PREPARE attributes string. You must provide the statement name and statement ID from IFCID 0316 output. For more information about using IFI to obtain information about the dynamic statement cache, see Monitoring the dynamic statement cache with READS calls.

0346 Monitors all invoked packages for a DB2 thread.

0369 Returns aggregated accounting statistics by connection type.

0373 Returns a pointer to the DSNHDECP load module or a user specified application defaults module that was loaded by the attached subsystem when DB2 was started.

0401 Returns information about static SQL statements that are tracked in the in the EDM pool. The statement identified can be used to query the catalog to identify the statement text. Data is accumulated when IFCID 0400 is activated. For more information about using IFI to obtain information about static SQL statements, see "Monitoring static SQL statements with READS calls" on page 826

Notes:

1. The following IFCID fields are available only through READS calls:

   0124

   0129

   0147

   0148

   0149

   0150

   0199

0234

0254

0316

For more information about IFCID field descriptions, see the mapping macros in the *prefix*.SDSNMACS data set.

PSPI

Related concepts:

DB2 trace

Related tasks:

Requesting data synchronously from a monitor program

Related reference:

READS

Trace field descriptions

Instrumentation facility interface (IFI) records

Qualification fields for READS requests

Monitor programs can use the *qualification area* of READS requests to specify constraints on the data that is to be returned by the request.

PSPI

If the qualification area does not exist (length of binary zero), information is obtained from all active allied threads and database access threads. Information is not obtained for any inactive database access threads that might exist.

Certain trace records cannot be qualified. Any qualifications are ignored for the following for the trace records that have the following IFCIDs: 0001, 0002, 0106, 0202, 0217, 0225, 0230

All other synchronous records can be qualified. However, certain trace records can use only certain qualifications fields. For information about the qualifications that particular trace records can use, see Table 177 on page 853.

Qualification area fields for READS requests

Unless the qualification area has a length of binary zero (in which case the area does not exist), the address of the qualification area supplied by the monitor program points to an area that is formatted by the monitor program, as shown in the following table.

Table 174. Qualification area fields. This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLEN	0	Signed two-byte integer	Length of the qualification area, plus 4. The following constants set the qualification area length field: • WQALLN9 (920 bytes) • WQALLN6 (264 bytes) • WQALLN5 (192 bytes) • WQALLN4 (168 bytes) • WQALLN21 (156 bytes) • WQALLN22 (120 bytes) • WQALLN23 (78 bytes) For information about the qualification fields that particular trace records can use, see Table 177 on page 853. For more information about the locations of qualification fields in the qualification area, see member DSNDWQAL of the *prefix*.SDSNMACS data set.
	2	Signed two-byte integer	Reserved.
WQALEYE	4	Character, 4 bytes	Eye catcher for block, WQAL.
WQALACE	8	Address	Thread identification token value. This value indicates the specific thread wanted; binary zero if it is not to be used.
WQALAIT2	C	Address	Reserved.
WQALPLAN	10	Character, 8 bytes	Plan name; binary zero if it is not to be used.
WQALAUTH	18	Character, 8 bytes	The current primary authorization ID; binary zero if it is not to be used.
WQALOPID	20	Character, 8 bytes	The original authorization ID; binary zero if it is not to be used.
WQALCONN	28	Character, 8 bytes	Connection name; binary zero if it is not to be used.
WQALCORR	30	Character, 12 bytes	Correlation ID; binary zero if it is not to be used.
WQALREST	3C	Character, 32 bytes	Resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or from a READS request for IFCID 0147 or 0148.
WQALHASH	5C	Hex, 4 bytes	Resource hash value that specifies the resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or possibly from a READS request for IFCID 0147 or 0148.
WQALASID	60	Hex, 2 bytes	ASID that specifies the address space of the process.
WQALFOPT	62	Hex, 1 byte	Filtering options for IFCID 0150: **X'80'** Return lock information for resources that have local or global waiters. (WQALLCON) **X'40'** Return lock information only for resources that have one or more interested agents. (WQALMAGN) **X'20'** Return lock information only for resources that have waiters. (WQALWAIT)

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFLGS	63	Hex, 1 byte	Options for 147/148 records: **X'40'** Active allied agent 147/148 records are **not** written for this READS request. DDF/RRSAF rollup records are written if WQAL148NR = OFF. (WQAL148NA) **X'80'** DDF/RRSAF rollup 147/148 records are **not** written for this READS request. Active allied agent records are written if WQAL148NA = OFF. [1] (WQAL148NR)
WQALLUWI	64	Character, 24 bytes	LUWID (logical unit of work ID) of the thread wanted; binary zero if it is not to be used
	7C	Character, 16 bytes	Location name. If specified, data is returned only for distributed agents that originate at the specified location. **Example:** If site A is located where the IFI program is running and SITE A is specified in the WQALLOCN, database access threads and distributed allied agents that execute at SITE A are reported. Local non-distributed agents are not reported. **Example:** If site B is specified in the WQALLOCN and the IFI program is still executing at site A, information on database access threads that execute in support of a distributed allied agent at site B are reported. **Example:** If WQALLOCN is not specified, information on all threads that execute at SITE A (the site where the IFI program executes) is returned. This includes local non-distributed threads, local database access agents, and local distributed allied agents.
WQALLTYP	8C	Character, 3 bytes	Specifies the type of log data access. 'CI ' must be specified to obtain log record control intervals (CIs).

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLMOD	8F	Character, 1 byte	The mode of log data access: **'D'** Retrieves the single log record whose RBA value and member id is specified in WQALLRBA. Issuing a D request while holding a position in the log, causes the request to fail and terminates the log position held. **'F'** Used as a first call to request log records beyond the LRSN or RBA specified in WQALLRBA that meet the criteria specified in WQALLCRI. **'H'** Retrieves the highest LRSN or log RBA in the active log. The value is returned in field IFCAHLRS of the IFI communications area (IFCA). There is no data returned in the return area and the return code for this call will indicate that no data was returned. **'N'** Used following mode F or N calls to request any remaining log records that meet the criteria specified in WQALLCRI. * and any option specified in WQALLOPT. As many log records as fit in the program's return area are returned. **'T'** Terminates the log position that was held by any previous F or N request. This allows held resources to be released.
WQALLNUM	90	Hex, 2 bytes	The number of log CIs to be returned. The valid range is X'0001' to X'0007'.
WQALCDCD	92	Character, 1 byte	Data description request flag: **'A'** Indicates that a data description can only be returned the first time a DATA request is issued from the region or when it was changed for a given table. This is the default. **'N'** Indicates that a data description is not returned. **'Y'** Indicates that a data description is returned for each table in the list for every new request.
	93	Hex, 1 byte	Reserved.
WQALLRBA	94	Hex, 8 bytes	If the IFCID is 0129, this is the starting log RBA of the CI to be returned. The CI starting log RBA value must end in X'000'. The RBA value must be right-justified. If the IFCID is 0306, this is the log RBA or LRSN to be used in mode 'F'.

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALGBPN	9C	Character, 8 bytes	Group buffer pool name for IFCID 0254. Buffer pool name for IFCID 0199. To specify a single buffer pool or group buffer pool, specify the buffer pool name in hexadecimal, followed by hexadecimal blanks. **Example:** To specify buffer pool BP1, put X'C2D7F14040404040' in this field. To specify more than one buffer pool or group buffer pool, use the pattern-matching character X'00' in any position in the buffer pool name. X'00' indicates that any character can appear in that position, and in all positions that follow. **Example:** If you put X'C2D7F10000000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP1, so IFI collects data for BP1, BP10 through BP19, and BP16K0 through BP16K9. **Example:** If you put X'C2D700F100000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP, so IFI collects data for all buffer pools. IFI ignores X'F1' in position four because it occurs after the first X'00'.
WQALLCRI	A4	Hex, 1 byte	Log Record Selection Criteria: Indicates what types of log records are returned: **X'00' (WQALLCR0)** Only log records for changed data capture and unit of recovery control. **X'FF' (WQALLCRA)** All types of log records. Use of this option can retrieve large data volumes and degrade DB2 performance.
WQALLOPT	A5	Hex, 1 byte	Processing options relating to decompression: **X'00'** Indicates that decompression should not occur. (WQALLOP0) **X'01'** Indicates to decompress the log records if they are compressed. (WQALLOP1)

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFLTR	A6	Hex, 1 byte	For IFCID 0316 requests, WQALFLTR identifies the filter method: **X'00'** Indicates no filtering. This value tells DB2 to return information for as many cached statements as fit in the return area. (WQALFLT0) **X'01'** Indicates that DB2 returns information about the cached statements that have the highest values for a particular statistics field. The statistics field is specified in WQALFFLD. DB2 returns information for as many statements as fit in the return area. (WQALFLT1) **Example:** If the return is large enough for information about 10 statements, the statements with the ten highest values for the specified statistics field are reported. **X'02'** Indicates that DB2 returns information about the cached statements that exceed a threshold value for a particular statistics field. The name of the statistics field is specified in WQALFFLD. The threshold value is specified in WQALFVAL. DB2 returns information for as many qualifying statements as fit in the return area. **X'04'** Indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0316 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock. For IFCID 0401 requests: **X'00'** Indicates no filtering. This value tells DB2 to return information for as many statements as fit in the return area. (WQALFLT0) **X'02'** Indicates that DB2 returns information about the statements that exceed a threshold value for a particular statistics field. The name of the statistics field is specified in WQALFFLD. The threshold value is specified in WQALFVAL. DB2 returns information for as many qualifying statements as fit in the return area. For IFCID 0317 requests, WQALFLTR identifies the filter method: **X'04'** Indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0317 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock. For IFCID 0306 requests, WQALFLTR indicates whether DB2 merges log records in a data sharing environment: **X'00'** Indicates that DB2 merges log records from data sharing members. (WQALMERG) **X'03'** Indicates that DB2 does not merge log records from data sharing members. (WQALNOMR)

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFFLD	A7	Character, 1 byte	For an IFCID 0316 request, when WQALFLTR is X'01' or X'02', this field specifies the statistics field that is used to determine the cached statements about which DB2 reports. You can enter the following values: **'A'** The accumulated elapsed time (QW0316AE). This option is valid only when QWALFLTR=X'01'. (WQALFFLA) **'B'** The number of buffer reads (QW0316NB). (WQALFFLB) **'C'** The accumulated CPU time (QW0316CT). This option is valid only when QWALFLTR=X'01'. (WQALFFLC) **'E'** The number of executions of the statement (QW0316NE). (WQALFFLE) **'G'** The number of GETPAGE requests (QW0316NG). (WQALFFG) **'I'** The number of index scans (QW0316NI). (WQALFFLI) **'L'** The number of parallel groups (QW0316NL). (WQALFFLL) **'P'** The number of rows processed (QW0316NP). (WQALFFLP) **'R'** The number of rows examined (QW0316NR). (WQALFFLR) **'S'** The number of sorts performed (QW0316NS). (WQALFFLS) **'T'** The number of table space scans (QW0316NT). (WQALFFLT) (Continued in the following row.)

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFFLD (continued)	A7	Character, 1 byte	(Continued from the previous row.) **'W'** The number of buffer writes (QW0316NW). (WQALFFLW) **'X'** The number of times that a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits (QW0316RT). **'Y'** The number of times that a RID list was not used because not enough storage was available (QW0316RS). (WQALFFLY) **'1'** The accumulated wait time for synchronous I/O (QW0316W1). This option is valid only when QWALFLTR=X'01'. (WQALFFL1) **'2'** The accumulated wait time for lock requests (QW0316W2). This option is valid only when QWALFLTR=X'01'. (WQALFFL2). **'3'** The accumulated wait time for a synchronous execution unit switch (QW0316W3). This option is valid only when QWALFLTR=X'01'. (WQALFFL3) **'4'** The accumulated wait time for global locks (QW0316W4). This option is valid only when QWALFLTR=X'01'.(WQALFFL4) **'5'** The accumulated wait time for read activity by another thread (QW0316W5). This option is valid only when QWALFLTR=X'01'. (WQALFFL5) **'6'** The accumulated wait time for write activity by another thread (QW0316W6). This option is valid only when QWALFLTR=X'01'. (WQALFFL6) **'7'** The accumulated wait time for lock requests (QW0316W7). This option is valid only when QWALFLTR=X'01'. (WQALFFL7). **'8'** The update timestamp when statistics were last changed (QW0316W8). This option is valid only when QWALFLTR=X'01'. (WQALFFL8). **'9'** The insertion timestamp when statements first enter the dynamic statement cache or EDM pool (QW0316W9). QWALFLTR=X'01'. (WQALFFL9)
WQALFVAL	A8	Signed 4-byte integer	For an IFCID 0316 request, when WQALFLTR is X'02', this field and WQALFFLD determine the cached statements about which DB2 reports. To be eligible for reporting, a cached statement must have a value for WQALFFLD that is no smaller than the value that you specify in WQALFVAL. DB2 reports information on as many eligible statements as fit in the return area.

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALSTNM	AC	Character, 16 bytes	For an IFCID 0317 request, when WQALFLTR is not X'04', this field specifies the name of a cached statement about which DB2 reports. This is a name that DB2 generates when it caches the statement. To obtain this name, issue a READS request for IFCID 0316. The name is in field QW0316NM. This field and WQALSTID uniquely identify a cached statement.
WQALSTID	BC	Unsigned 4-byte integer	For an IFCID 0316 or IFCID 0317 request, this field specifies the ID of a cached statement about which DB2 reports. DB2 generates this ID when it caches the statement. To obtain the ID, use the following options: • For an IFCID 0317 request, when WQALFLTR is not X'04', obtain this ID by issuing a READS request for IFCID 0316. The ID is in field QW0316TK. This field and WQALSTNM uniquely identify a cached statement. • For an IFCID 0316 or IFCID 0317 request, when WQALFLTR is X'04', obtain this ID by issuing a READS request for IFCID 0172 or IFCID 0196. The ID is in field QW0172H9 (cached statement ID for the holder in a deadlock), QW0172W9 (cached statement ID for the waiter in a deadlock), or QW0196H9 (cached statement ID of the holder in a timeout). This field uniquely identifies a cached statement.
WQALEUID	C0	Character, 16 bytes	The first 16 bytes of the user's workstation user ID. This value can be different from the authorization ID that is used to connect to DB2.
WQALEUTX	D0	Character, 32 bytes	The first 32 bytes of the name of the transaction or application that the user is running. This value identifies the application that is running, not the product that is used to run the application.
WQALEUWN	F0	Character, 18 bytes	The first 18 bytes of the user's workstation name. This value can be different from the authorization ID used to connect to DB2.
WQALPLOC	102	Character, 128 bytes	The package location name.
WQALPCOL	182	Character, 128 bytes	The collection name of the package that the user is running.
WQALPPNM	202	Character, 128 bytes	The program name of the package that the user is running.
WQALROLE	282	Character, 128 bytes	The connection role of the user. This field contains binary zeros if the client does not supply this information.
WQALRTBL	302	Character, 128 Bytes	The name of the referenced table.

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALEXCD	382	Character, 4 bytes	This field specifies the level of exclude filtering to be performed, according to the following values: **X'80000000'** Specifies exclude filtering by plan name (WQALPLAN) **X'40000000'** Specifies exclude filtering by current authorization ID (WQALAUTH) **X'20000000'** Specifies exclude filtering by original authorization ID (WQALOPID) **X'10000000'** Specifies exclude filtering by connection name (WQALCONN) **X'08000000'** Specifies exclude filtering by correlation name (WQALCORR) **X'04000000'** Specifies exclude filtering by cached statement ID (WQALASID) **X'02000000'** Specifies exclude filtering by location name (WQALLUWI) **X'01000000'** Specifies exclude filtering by location name (WQALLOCN) **X'00800000'** Specifies exclude filtering by workstation ID (WQALEUID) **X'00400000'** Specifies exclude filtering by transaction or application name (WQALEUTX) **X'00200000'** Specifies exclude filtering by workstation name (WQALEUWN) **X'00100000'** Specifies exclude filtering by package location name (WQALEPLOC) **X'00080000'** Specifies exclude filtering by package collection name (WQALEPCOL) **X'00040000'** Specifies exclude filtering by package program name (WQALEPPNM) **X'00002000'** Specifies exclude filtering by connection role (WQALEROLE)
	386	Character, 2 bytes	Reserved.

Table 174. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFVAL64	388	Signed integer, 8 bytes	64-bit threshold value for IFCID 0316 and 0401 requests. If the value is non-zero and the target system is DB2 10 or higher, this value is used in place of WQALFVAL.
WQALWQLS	390	Address, 4 bytes	Reserved for future. Must be 0.
WQALWQL4	394	Address, 4 bytes	Address of WQLS or 0.

Notes:

1. The only valid filters for DDF/RRSAF 147/148 rollup records are WQALEUID, WQALEUTX, and WQALEUWN. For a 147/148 request, DDF/RRSAF records are not processed if any of the following WQAL fields are not X'00':
   - WQALACE
   - WQALAUTH
   - WQALOPID
   - WQALPLAN
   - WQALCORR
   - WQALLUWI
   - WQALLOCN
   - WQALASID
   - WQALCONN
   - WQALPLOC
   - WQALPCOL
   - WQALPPNM
   - WQALROLE

WQLS maps the area used to filter IFCID 0306 records. Unless the qualification area has a length of binary zero (in which case the area does not exist), the address of the qualification area supplied by the monitor program points to an area that is formatted by the monitor program, as shown in the following table.

Table 175. Qualification area fields of WQALWQLS. This area is mapped by the assembler mapping macro DSNDWQAL. This area maps the area used to filter IFCID 0306 records.

Name	Hex offset	Data type	Description
WQLSLEN	0	Signed integer, 4 bytes	Length of WQLS.
WQLSEYE	4	Character, 2 bytes	Eye catcher for block, WQLS.
reserved	8	Character, 48 bytes	Reserved
WQLSTYPE	38	Character, 4 bytes	Type of qualification items
WQLSITEM	3C	Signed integer, 4 bytes	Number of qualification items
WQLSLIST	40	Array	List of qualification items in WQLSDBPS.

WQLSDBPS is located at offset X'40' in WQLS.

Table 176. Qualification area fields of WQLSDBPS. This area is mapped by the assembler mapping macro DSNDWQAL. This area maps the area used to filter IFCID 0306 records when WQLSTYPE='DBPS'.

Name	Hex offset	Data type	Description
WQLSDBID	0	Hex, 2 bytes	DBID of database.
WQLSPSID	2	Hex, 2 bytes	PSID of table space or OBID of table for ALTER TABLE diagnostic log records.

If your monitor program does not initialize the qualification area, the READS request is denied.

Qualification fields that are used by IFCIDs

The following table lists the qualification fields that can be used for certain IFCIDs that contain synchronous data. For descriptions of these IFCIDs, see "Trace fields for READS requests" on page 839

Table 177. Qualification fields that are used by particular IFCIDs

These IFCIDs	Are allowed to use these qualification fields	Minimum length (WQALLEN constant) to use all qualifications [3]
0001, 0002, 0104, 0106, 0202, 0217, 0225, 0230, 0373	None. All qualifications are ignored.	Not applicable.
0124, 0147, 0148, 0150, 0346	WQALACE WQALAIT2 WQALPLAN[1] WQALAUTH[1] WQALOPID[1] WQALCONN[1] WQALCORR[1] WQALASID WQALLUWI[1] WQALLOCN[1] WQALEUID WQALEUTX WQALEUWN WQALPLOC WQALPCOL WQALPPNM WQALROLE[1]	WQALLN9
0129	WQALLTYP WQALLMOD WQALLRBA WQALLNUM	WQALLN23
0149	WQALREST WQALHASH	WQALLN6
0150	WQALFOPT	WQALLN21
0185	WQALCDCD	WQALLN4
0199, 0254	WQALGBPN[2]	WQALLN5

Table 177. Qualification fields that are used by particular IFCIDs (continued)

These IFCIDs	Are allowed to use these qualification fields	Minimum length (WQALLEN constant) to use all qualifications [3]
0306	WQALFLTR WQALLMOD WQALLRBA WQALLCRI WQALLOPT WQALWQLS	WQALLN23
0316	WQALFLTR WQALFFLD WQALFVAL WQALFVAL64 WQALSTID Filtering by the following fields apply, only after all other filters for IFCID 0316 are applied: WQALEUID WQALEUTX WQALEUWN WQALRTBL	WQALLN9
0317	WQALFLTR WQALSTNM WQALSTID	WQALLN5
0401	WQALFLTR WQALFFLD WQALFVAL WQALFVAL64	WQALLN9

Notes:

1. DB2 allows you to partially qualify a field and fill the rest of the field with binary zero. For example, the 12-byte correlation value for a CICS thread contains the 4-character CICStransaction code in positions 5-8. Assuming a CICS transaction code of AAAA, the following hexadecimal *qual-area* correlation qualification can be used to find the first transaction with a correlation value of AAAA in positions 5-8: X'00000000C1C1C1C100000000'.
2. X'00' in this field indicates a pattern-matching character. X'00' in any position of the field indicates that IFI collects data for buffer pools whose names contain any character in that position and all following positions.
3. The specified WQALLEN constant represents the minimum qualification area length that is required for the use of all qualifications for the specified IFICIDs. A longer length can also be specified for any IFCID. However, the length must match one of these constants.

PSPI

Related concepts:

DB2 trace

DB2 trace output

Related reference:

READS

Trace field descriptions

WRITE

Monitor programs can call the WRITE function to write data to the instrumentation facility interface.

Authorization

PSPI

WRITE requests are not checked for authorization, but a DB2 trace must be active for the IFCID being written. If the IFCID is not active, the request is denied. For a WRITE request, no other authorization checks apply.

Syntax and options

WRITE calls to the instrumentation facility interface have the following syntax:

```
CALL DSNWLI,('WRITE   ',ifca,output-area,ifcid-area),VL
```

The write function must specify an IFCID area. The data that is written is defined and interpreted by your site.

ifca
Contains information regarding the success of the call.

output-area
Contains the varying-length of the monitor program's data record to be written.

ifcid-area
Contains the IFCID of the record to be written. Only the IFCIDs that are shown in the following table are supported by the WRITE function. If an invalid IFCID is specified or the IFCID is not active (not started by a TRACE command), no data is written.

Table 178. Valid IFCIDs for WRITE function

IFCID (decimal)	IFCID (hex)	Trace type	Class	Comment
0146	0092	Auditing	9	Write to IFCID 146
0151	0097	Accounting	4	Write to IFCID 151
0152	0098	Statistics	2	Write to IFCID 152
0153	0099	Performance	1	Background events and write to IFCID 153
0154	009A	Performance	15	Write to IFCID 154
0155	009B	Monitoring	4	Write to IFCID 155
0156	009C	Global (Serviceability)	6	Reserved for user-defined serviceability trace

PSPI

Related tasks:

Writing to trace destinations from monitor programs

Related reference:

Instrumentation facility communications area (IFCA)
IFCID area
Output area

Common communication areas for IFI calls

The following communication areas are used on all IFI calls.

Instrumentation facility communications area (IFCA)

A monitor program's *instrumentation facility communication area (IFCA)* is a communications area between the monitor program and IFI. The IFCA contains information about the success of the call in its return code and reason code fields. The IFCA is a required parameter on all IFI requests.

PSPI

The monitor program is responsible for allocating storage for the IFCA and initializing it. The IFCA must be initialized to binary zeros and the eye catcher, 4-byte owner field, and length field must be set by the monitor program. Failure to properly initialize the IFCA results in denying any IFI requests.

The monitor program is also responsible for checking the IFCA return code and reason code fields to determine the status of the request.

The IFCA fields are described in the following table.

Table 179. Instrumentation facility communication area. The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCALEN	0	Hex, 2 bytes	Length of IFCA.
IFCAFLGS	2	Hex, 1 byte	Processing flags. • IFCAGLBL, X'80' This bit is on if an IFI request is to be processed on all members of a data sharing group. • IFCASYNC, X'40' This bit is on when IFI run commands synchronously, for commands that can be run synchronously or asynchronously
	3	Hex, 1 byte	Reserved.
IFCAID	4	Character, 4 bytes	Eye catcher for block, IFCA.
IFCAOWNR	8	Character, 4 bytes	Owner field, provided by the monitor program. This value is used to establish ownership of an OP*n* destination and to verify that a requester can obtain data from the OP*n* destination. This value is not the same as the owner ID of a plan.
IFCARC1	C	4-byte signed integer	Return code for the IFI call. Binary zero indicates a successful call. For information about other return codes, see X'E6......' codes (DB2 Codes). For a return code of 8 from a COMMAND request, the IFCAR0 and IFCAR15 values contain additional information.

Table 179. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCARC2	10	4-byte signed integer	Reason code for the IFI call. Binary zero indicates a successful call. For information about other reason codes, see X'E6......' codes (DB2 Codes).
IFCABM	14	4-byte signed integer	Number of bytes moved to the return area. A non-zero value in this field indicates information was returned from the call. Only complete records are moved to the monitor program area.
IFCABNM	18	4-byte signed integer	Number of bytes that did not fit in the return area and still remain in the buffer. Another READA request retrieves that data. Certain IFI requests return a known quantity of information. Other requests terminate when the return area is full.
	1C	4-byte signed integer	Reserved.
IFCARLC	20	4-byte signed integer	Indicates the number of records lost prior to a READA call. Records are lost when the OP buffer storage is exhausted before the contents of the buffer are transferred to the application program via an IFI READA request. Records that do not fit in the OP buffer are not written and are counted as records lost.
IFCAOPN	24	Character, 4 bytes	Destination name used on a READA request. This field identifies the buffer requested, and is required on a READA request. Your monitor program must set this field. The instrumentation facility fills in this field on START TRACE to an OP*n* destination from an monitor program. If your monitor program started multiple OP*n* destination traces, the first one is placed in this field. If your monitor program did not start an OP*n* destination trace, the field is not modified. The OP*n* destination and owner ID are used on subsequent READA calls to find the asynchronous buffer.
IFCAOPNL	28	2-byte signed integer	Length of the OP*n* destinations started. On any command entered by IFI, the value is set to X'0004'. If an OP*n* destination is started, the length is incremented to include all OP*n* destinations started.
	2A	2-byte signed integer	Reserved.
IFCAOPNR	2C	Character, 8 fields of 4 bytes each	Space to return 8 OP*n* destination values.
IFCATNOL	4C	2-byte signed integer	Length of the trace numbers plus 4. On any command entered by IFI the value is set to X'0004'. If a trace is started, the length is incremented to include all trace numbers started.
	4E	2-byte signed integer	Reserved.
IFCATNOR	50	Character, 8 fields of 2 bytes each.	Space to hold up to eight EBCDIC trace numbers that were started. The trace number is required if the MODIFY TRACE command is used on a subsequent call.
IFCADL	60	Hex, 2 bytes	Length of diagnostic information.
	62	Hex, 2 bytes	Reserved.

Table 179. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCADD	64	Character, 80 bytes	Diagnostic information. • IFCAFCI, offset 64, 6 bytes This contains the RBA of the first CI in the active log if IFCARC2 is 00E60854. • IFCACR0, offset 64, 4 bytes For COMMAND requests, this field contains -1 or the return code from the component that executed the command. • IFCACR15, offset 68, 4 bytes [1] • IFCAR0, offset 6C, 4 bytes Log manager reason code for IFCID 0129 requests, if IFCARC2 is 00E60834, 00E60835, or 00E60836. • IFCAR15, offset 70, 4 bytes Log manager return code for IFCID 0129 requests, if IFCARC2 is 00E60834, 00E60835, or 00E60836. • IFCAR15 offset 70, 4 bytes • IFCAGBPN, offset 74, 8 bytes This is the group buffer pool name in error if IFCARC2 is 00E60838 or 00E60860 • IFCABSRQ, offset 88, 4 bytes – If the reason code is 00E60864, this value is the size of the return area that is required. • IFCAHLRS, offset 8C, 6 bytes This field contains one of the following values: – If WQALLMOD='H', the highest LRSN or log RBA in the active log. – If WQALLMOD=L', the difference between the current and previous LRSN for the system
IFCAGRSN	98	Four-byte signed integer	Reason code for the situation in which an IFI calls requests data from members of a data sharing group, and not all the data is returned from group members.
IFCAGBM	9C	Four-byte signed integer	Total length of data that was returned from other data sharing group members and fit in the return area.
IFCAGBNM	A0	Four-byte signed integer	Total length of data that was returned from other data sharing group members and did not fit in the return area.
IFCADMBR	A4	Character, 8 bytes	Name of a single data sharing group member on which an IFI request is to be executed. Otherwise, this field is blank. If this field contains a member name, DB2 ignores field IFCAGLBL.
IFCARMBR	AC	Character, 8 bytes	Name of the data sharing group member from which data is being returned. DB2 sets this field in each copy of the IFCA that it places in the return area, not in the IFCA of the application that makes the IFI request.

Notes:

1. For COMMAND requests, the IFCACR15 field contains one of the following values:

   0 The command completed successfully.

   4 Internal error.

8 The command was not processed because of errors in the command.

12 The component that executed the command returned the return code in IFCAR0.

16 An abend occurred during command processing. Command processing might be incomplete, depending on when the error occurred. See IFCAR0 for more information.

20 Response buffer storage was not available. The command completed, but no response messages are available. See IFCAR0 for more information.

24 Storage was not available in the DSNMSTR address space. The command was not processed.

28 CSA storage was not available. If a response buffer is available, the command might have partially completed. See IFCAR0 for more information.

32 The user is not authorized to issue the command. The command was not processed.

PSPI

Related concepts:

Errors and IFI

Related tasks:

Reading specific log records (IFCID 0129) (DB2 Administration Guide)

Specifying the return area (DB2 Administration Guide)

Related reference:

IFI functions

Return area

You must specify a return area on all READA, READS, and command requests.

PSPI

IFI uses the return area to return command responses, synchronous data, and asynchronous data to the monitor program. Table 180describes the return area.

Table 180. Return area

Hex offset	Data type	Description
0	Signed 4-byte integer	The length of the return area, plus 4. This must be set by the monitor program. No limit exists for the length of READA or READS return areas.

Table 180. Return area (continued)

Hex offset	Data type	Description
4	Character, varying-length	DB2 places as many varying-length records as it can fit into the area following the length field. The monitor program's length field is not modified by DB2. Each varying-length trace record has either a 2-byte or 4-byte length field, depending on the high-order bit. If the high-order bit is on, the length field is 4 bytes. If the high-order bit is off, the length field is 2 bytes. In this case, the third byte indicates whether the record is spanned, and the fourth byte is reserved. After a command request, the last character in the return area is a new-line character (X'15').

The following table shows the return area for IFCID 0306.

Table 181. Return area using IFCID 0306

Hex offset	Data type	Description
0	Signed four-byte integer	The length of the return area.
4	Character, 4 bytes	The eye catcher, a constant, I306. Beginning of QW0306OF mapping.
8	Character, 128 bytes	Reserved.
88	Signed four-byte integer	The length of the returned data.

The destination header for data that is returned on a READA or READS request is mapped by macro DSNDQWIW or the header QW0306OF for IFCID 306 requests. Please refer to *prefix*.SDSNIVPD(DSNWMSGS) for the format of the trace record and its header. The size of the return area for READA calls should be as large the size specified with the BUFSIZE keyword on the START TRACE command.

Data returned on a command request consists of varying-length segments (X'xxxxrrrr' where the length is 2 bytes and the next 2 bytes are reserved), followed by the message text. More than one record can be returned. The last character in the return area is a new-line character (X'15').

The monitor program must compare the number of bytes moved (IFCABM in the IFCA) to the sum of the record lengths to determine when all records have been processed.

PSPI

Related information:

Reading log records (DB2 Administration Guide)

IFCID area

You must specify the IFCID area on READS and WRITE requests.

PSPI

The IFCID area contains the IFCIDs to process. The following table shows the IFCID area.

Table 182. IFCID area

Hex Offset	Data type	Description
0	Signed two-byte integer	Length of the IFCID area, plus 4. The length can range from X'0006' to X'0044'. For WRITE requests, only one IFCID is allowed, so the length must be set to X'0006'. For READS requests, you can specify multiple IFCIDs. If so, you must be aware that the returned records can be in a different sequence than requested and some records can be missing.
2	Signed two-byte integer	Reserved.
4	Hex, *n* fields of 2 bytes each	The IFCIDs to be processed. Each IFCID is placed contiguous to the previous IFCID for a READS request. The IFCIDs start at X'0000' and progress upward. You can use X'FFFF' to signify the last IFCID in the area to process.

PSPI

Output area

The output area is used on command and WRITE requests

PSPI

. The first two bytes contain the length of the monitor program's record to write or the DB2 command to be issued, plus 4 additional bytes. The next two bytes are reserved. You can specify any length from 10 to 4096 (X'000A0000' to X'10000000'). The rest of the area is the actual command or record text.

Example: In an assembler program, a START TRACE command is formatted in the following way:

```
DC   X'002A0000'       LENGTH INCLUDING LL00 + COMMAND
DC   CL37'-STA TRACE(MON) DEST(OPX) BUFSIZE(256)'
```

PSPI

Instrumentation facility interface (IFI) records

This information describes the format of the records returned by the instrumentation facility interface (IFI) as a result of READA, READS, and COMMAND requests.

Related concepts:

DB2 trace

Related tasks:

Requesting data synchronously from a monitor program

Requesting data asynchronously from a monitor program

Related reference:

COMMAND

IFCID Record Blocks (Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS)

Trace data record format

Trace records that are returned from READA and READS requests contain the following sections:

PSPI

- A writer header that reports the length of the entire record, whether the record was in the first, middle, or last section of data, and other specific information for the writer.
  - The writer header for IFI is mapped by DSNDQWIW or the header QW0306OF for IFCID 306 requests. See the mapping macros in *prefix*.SDSNMACS for the formats.
  - In SMF, writer headers for statistics records are mapped by macro DSNDQWST, for accounting records by DSNDQWAS, and for performance, audit, and monitor records by DSNDQWSP. When these macros are assembled, they include the other macros necessary to map the remainder of the trace records sent to SMF. The length and content of the writer header section differs between SMF and GTF records, but the other sections of the records are the same for SMF and GTF.
  - The GTF writer header section begins at the first byte of the record. After establishing addressability, you can examine the fields of the header. The writer headers for trace records sent to GTF are always mapped by macro DSNDQWGT. The fields are described in Table 166 on page 803.
- A self-defining section
- A product section containing specific DB2 information based on the active trace. The product section for all record types contains the standard header, DSNDQWHS. The other headers (correlation, CPU, distributed, and data sharing data) might also be present.

  DSNDQWHC
  Product section correlation header

  DSNDQWHD
  Distributed data header

  DSNDQWHT
  Trace header

  DSNDQWHA
  Data sharing header

  DSNDQWHU
  CPU header

  DSNWMSGS.
  Descriptive text for all IFCID records
- Data areas containing the actual recorded data are mapped by multiple mapping macros described in *prefix*.SDSNMACS.

The following figure shows the return area after a READS request successfully executed.

```
DFSERA10 - PRINT PROGRAM
⋮
          A        B                  C
 000000   05A80000 00000510  00980001 00000054   00B80001 0000010C  01000001 0000020C
 000020   01160001 00000324  01B00001 000004D4   00000000 000004D4  00080001 000004DC
                                                          D
```

```
000040   00010001 000004E0  00000000 000004E0   00300001 80000018  00000010 000003E8
000060   00640064 000A0028  003D0000 0000A000   00033000 00033000  00010000 E0000000
000080   00000000 00000000  00000000 C1C4D4C6   F0F0F140 F0F20080  00003084 00000000
0000A0   00002000 0005003C  0028F040 40404040   40404040 40404040  40404040 40404040

⋮
000320   B0000000 202701D0  E2D7D9D4 D2C4C4F0   F0F1F940 01980064  00000000 E7C14000
000340   00400280 C5E2E8E2  C1C4D440 40000000   000E1000 000001BC  000001B0 C9C2D4E4
000360   E2C5D940 C9D9D3D4  D7D9D6C3 C9D9D3D4   0000003C 0000012C  0000000A 8080008C
000380   00FA0000 00007D00  000A0014 00050028   000E0002 00080008  00400077 00000514
0003A0   000003E8 012C0000  0000000E 000A01F4   00FA0000 00000032  000003E8 00002710
0003C0   E2E8E2C1 C4D44040  E2E8E2D6 D7D94040   E2E8E2D6 D7D94040  000A0080 00140000
0003E0   00000080 0005000A  13880078 0008000A   00040004 00040005  0001000A 00020005
000400   00003000 00007800  00000001 000007D0   00040400 00780078  00010003 00019000
000420   0000000A 00000020  00000019 00000000   0005000A 0006000A  00640064 00040063
000440   00000000 00000000  00000000 00000000   00000000 00000000  00000000 00000000
000460   30C4E2D5 D9C7C3D6  D3C4E2D5 6DD9C5C7   C9E2E3C5 D96DC1D7  D7D3C4E2 D56DD9C5
000480   C7C9E2E3 C5D96DD6  C2D1E380 C4E2D5D9   C7C6C4C2 000009FD  C5000000 00001060
0004A0   00020000 00001000  40000000 00000000   00000000 00000000  00000000 00000000
0004C0   00000000 00000000  00000000 F1F161F1   F361F9F2 C4E2D5C3  F3F1F040 80000000
0004E0   00160030 C6C1C340  00010000 C4C4C640   40404040 C1800002  00000000 C1C3E3C9
                                                E        F
000500   E5C54040 00000000  00000000 00000000   004C011A 006A0A31  00B45B78 E2E2D6D7
000520   A6E9C7D5 EBDB1104  00000008 00000002   00000001 E2C1D5E3  C16DE3C5 D9C5E2C1
000540   6DD3C1C2 C4C2F2D5  C5E34040 D3E4D5C4   F0404040 A6E9C7D2  E73C0001 004C0200
000560   E2E8E2C1 C4D44040  D4D7C9E3 E2F14040   40404040 C2C1E3C3  C8404040 C4E2D5C5
000580   C4C3D340 E2E8E2C1  C4D44040 00000001   00000000 00000000  00000000 00000000
0005A0   00000000 00000000
```

Figure 70. Example of IFI return area after READS request (IFCID 106). This output was assembled by a user-written routine and printed with the DFSERA10 print program of IMS.

Figure label	Description
A 05A8	Length of record. The next two bytes are reserved.
B 00000510	Offset to product section standard header.
C 00000054	Offset to first data section.
D 80000018	Beginning of first data section.
E 004C011A	Beginning of product section standard header.
F 006A	IFCID (decimal 106).

PSPI

Related concepts:

DB2 trace output

DB2 trace

Related reference:

Product section

Command record format

The record that is returned from a command request can contain none or many message text segments.

PSPI

Each segment is a varying-length message (LLZZ, where LL is the 2-byte length and ZZ is a 2-byte reserved area) followed by message text. The IFCA's IFCABM field contains the total number of bytes moved.

The following figure shows the return area after a START TRACE command successfully executes.

```
DFSERA10 - PRINT PROGRAM
⋮
          A        B         C        D
 000000   007E0000 0000007A  003C0001 C4E2D5E6   F1F3F0C9 406F40D4  D6D540E3 D9C1C3C5
 000020   40E2E3C1 D9E3C5C4  6B40C1E2 E2C9C7D5   C5C440E3 D9C1C3C5  40D5E4D4 C2C5D940
                   E         F
 000040   F0F24015 003A0001  C4E2D5F9 F0F2F2C9   406F40C4 E2D5E6E5  C3D4F140 7D60E2E3
 000060   C1D9E340 E3D9C1C3  C57D40D5 D6D9D4C1   D340C3D6 D4D7D3C5  E3C9D6D5 4015
```

Figure 71. Example of IFI return area after a START TRACE command. This output was assembled with a user-written routine and printed with DFSERA10 program of IMS

Figure label	Description
A 007E0000	Field entered by print program
B 0000007A	Length of return area
C 003C	Length of record (003C). The next two bytes are reserved.
D C4E2D5E6	Beginning of first message
E 003A	Length of record. The next two bytes are reserved.
F C4E2D5F9	Beginning of second message

The IFCABM field in the IFCA would indicate that X'00000076' (C + E) bytes have been moved to the return area.

PSPI

Part 10. Testing DB2 performance

You can take measures to ensure that your test subsystem is a realistic model of your production environment.

Related tasks:

➡ Testing and debugging an application program on DB2 for z/OS (DB2 Application programming and SQL)

Chapter 48. Modeling a production environment on a test subsystem

You can improve the accuracy of access path testing by modeling the configuration and settings of a production environment in a test subsystem. The test system uses values that you specify for processor configuration, RID pool, sort pool, and buffer pool settings.

Before you begin

- A complete set of EXPLAIN tables must exist with the SYSIBM qualifier on the production subsystem and in the test subsystem.

  You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.
- A complete set of profile tables must exist on the test subsystem.

  The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

About this task

Application testing is often conducted on test subsystems that have different parameters and configurations than the production subsystems that actually run the applications. Such differences can result in different access paths between the test and production subsystems. Such differences can cause performance problems to remain undetected on the test system, to be discovered only in the production environment.

However, you can specify that DB2 models the configuration and parameters of your production environment in your test subsystem. You can specify the following parameters and configuration details for DB2 to use for access path selection in your test subsystem:

- Processor speed
- Number of processors
- Maximum number of RID blocks
- Sort pool size
- Buffer pool sizes

Procedure

To model your production environment in a test subsystem:

1. Gather the values to model in the test system from your production system. The information that you might gather includes the processor speed and the number of processors, RID pool, sort pool, and buffer pool settings.

   a. Capture EXPLAIN information about the production system to make the needed values available. You can execute the EXPLAIN statement for any SQL statement, but you must specify a unique QUERYNO value. For example, you might execute the following statements:

   ```
   SET CURRENT DEGREE='ANY';
   EXPLAIN ALL SET QUERYNO=2647 FOR SELECT * FROM SYSIBM.SYSDUMMY1;
   ```

   b. Execute a query to gather the required values from the EXPLAIN data in the PLAN_TABLE. For example, you might use the following statements to gather the following information from the IBM_SERVICE_DATA column:

      - Processor speed
      - Number of processors
      - Maximum number of RID blocks
      - Sort pool size

   ```
   SELECT (CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),1,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),1,1)) END
         * POWER (16, 7)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),2,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),2,1)) END
    * POWER (16, 6)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),1,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),1,1)) END
    * POWER (16, 5)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),2,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),2,1)) END
    * POWER (16, 4)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),1,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),1,1)) END
    * POWER (16, 3)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),2,1)
   ```

```
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),2,1)) END
 * POWER (16, 0)) AS CPUSPEED_EXP,
 (CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),1,1)
      WHEN 'A' THEN 10
      WHEN 'B' THEN 11
      WHEN 'C' THEN 12
      WHEN 'D' THEN 13
      WHEN 'E' THEN 14
      WHEN 'F' THEN 15
    ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),1,1)) END
       * POWER (16, 3)
  + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),2,1)
      WHEN 'A' THEN 10
      WHEN 'B' THEN 11
      WHEN 'C' THEN 12
      WHEN 'D' THEN 13
      WHEN 'E' THEN 14
      WHEN 'F' THEN 15
    ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),2,1)) END
  * POWER (16, 2)
  + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),1,1)
      WHEN 'A' THEN 10
      WHEN 'B' THEN 11
      WHEN 'C' THEN 12
      WHEN 'D' THEN 13
      WHEN 'E' THEN 14
      WHEN 'F' THEN 15
    ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),1,1)) END
  * POWER (16, 1)
  + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),2,1)
      WHEN 'A' THEN 10
      WHEN 'B' THEN 11
      WHEN 'C' THEN 12
      WHEN 'D' THEN 13
      WHEN 'E' THEN 14
      WHEN 'F' THEN 15
    ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),2,1)) END
 * POWER (16, 0)) AS NUMCP_EXP,
(CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),1,1)) END
     * POWER (16, 7)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),2,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),2,1)) END
* POWER (16, 6)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),1,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),1,1)) END
* POWER (16, 5)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),2,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),2,1)) END
* POWER (16, 4)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),1,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),1,1)) END
* POWER (16, 3)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),2,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),2,1)) END
* POWER (16, 2)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),1,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),1,1)) END
* POWER (16, 1)
+ CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),2,1)
    WHEN 'A' THEN 10
    WHEN 'B' THEN 11
    WHEN 'C' THEN 12
    WHEN 'D' THEN 13
    WHEN 'E' THEN 14
    WHEN 'F' THEN 15
  ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),2,1)) END
* POWER (16, 0)) AS RIDPOOL_EXP,
(CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),1,1)) END
      * POWER (16, 7)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),2,1)) END
 * POWER (16, 6)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),1,1)) END
 * POWER (16, 5)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),2,1)) END
 * POWER (16, 4)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),1,1)) END
 * POWER (16, 3)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
```

```
  WHEN 'D' THEN 13
  WHEN 'E' THEN 14
  WHEN 'F' THEN 15
 ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),2,1)) END
* POWER (16, 0)) AS SORTPOOL_EXP
FROM PLAN_TABLE WHERE QUERYNO=2647;
```

   c. Use the DISPLAY BUFFERPOOL command to gather information about the buffer pools that are defined in the production environment.
   d. Gather information about the values of the following optimization subsystem parameters in the test environment:
      - NPGTHRSH
      - PARAMDEG
      - STARJOIN

      You can use either the DSNWZP stored procedure or the ADMIN_INFO_SYSPARM stored procedure to obtain the current settings of most documented DB2 subsystem parameters. The DSNTEJ6Z sample job contains a sample call to the DSNWZP stored procedure.
2. Set subsystem parameters on the test subsystem to simulate the processor speed and number of processors in the production environment.
   a. Modify a copy of the DSNTIJUZ job that you use to create the subsystem parameter module for the test subsystem, and add the following keyword parameters:

      SIMULATED_CPU_SPEED
      Specify the value that was captured from the production subsystem as CPUSPEED_EXP.

      SIMULATED_CPU_COUNT
      Specify the value that was captured from the production subsystem as NUMCP_EXP. This subsystem parameter value is applicable only for queries that use that use parallelism.

      NPGTHRSH
      Specify the same value that is used in the production environment.

      PARAMDEG
      Specify the same value that is used in the production environment.

      STARJOIN
      Specify the same value that is used in the production environment.
3. Create a profile on the test system to specify RID pool, sort pool, and buffer pool settings. The single profile that you create has a global scope for the single subsystem that it applies to.
   a. Create the profile. You can specify any unique value for PROFILEID.

```
INSERT INTO SYSIBM.DSN_PROFILE_TABLE (PROFILEID)
 VALUES (4713);
```

   b. Insert values into the DSN_PROFILE_ATTRIBUTES table to model production buffer pools in the test environments. For example, the following statements mean that DB2 uses a value of 25000 for BP0 and for a value of 2500 for BP8K. The values override the actual buffer pool sizes only when DB2 when determines access paths:

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
 (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
 VALUES
 (4713, 'BP0',NULL, 25000);
```

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
 (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
 VALUES
 (4713, 'BP8K0',NULL, 2500);
```

The actual buffer pool sizes are not changed on the test subsystem. The buffer pool assignment for each table in your test system must match the buffer pool assignment for the corresponding table in the production environment. However, it is unnecessary that the buffer pools of the same name be assigned to corresponding table in each environment. However, each table must be assigned to a buffer pool of the same size in each environment.

c. Insert values into the DSN_PROFILE_ATTRIBUTES table on the test to model the size of the sort pool on the production subsystem: Specify the value that was captured from the production subsystem as SORTPOOL_EXP.

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
 (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
 VALUES
 (4713, 'SORT_POOL_SIZE',NULL, 307200);
```

d. Insert values into the DSN_PROFILE_ATTRIBUTES table on the test subsystem to model the maximum number of RID blocks on the production subsystem: Specify the value that was captured from the production subsystem as RIDPOOL_EXP.

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
 (PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
 VALUES
 (4713, 'MAX_RIDBLOCKS',NULL, 300);
```

4. Issue a START PROFILE command to start the profile on the test subsystem. You must have SYSOPR, SYSCTRL, or SYSADM authority to issue the command. After you start the profile, you might execute the following statements to verify that the profile is started:

```
SELECT PROFILE_ENABLED FROM SYSIBM.DSN_PROFILE_TABLE
WHERE PROFILEID=4713;

SELECT SUBSTR(KEYWORDS,1,14) KEYWORDS,ATTRIBUTE2,
SUBSTR(STATUS,1,51) STATUS
FROM SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY PAH
WHERE PAH.ATTRIBUTE_TIMESTAMP =
(SELECT MAX(ATTRIBUTE_TIMESTAMP)
 FROM SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY PAH2
 WHERE PAH2.PROFILEID = PAH.PROFILEID)
AND PROFILEID = 4713;
```

If the profile is started, the preceding statements return the following result:

Table 183. Sample result from the DSN_PROFILE_ATTRIBUTES_HISTORY table

KEYWORDS	ATTRIBUTE2	STATUS
MAX_RIDBLOCKS	300	ACCEPTED
SORT_POOL_SIZE	307200	ACCEPTED
BP8K0	2500	ACCEPTED
BP0	25000	ACCEPTED

5. Validate that the correct parameters settings are used in the test system.

a. Capture EXPLAIN information in the test system to gather values that you can compare to the production environment. You might use either of the following methods to capture the information.

   - Issue an EXPLAIN ALL statement, such as the statement shown in the following example:

     ```
     SET CURRENT DEGREE='ANY';
     EXPLAIN ALL SET QUERYNO = 1 FOR
     SELECT COUNT(*) FROM SYSIBM.DSN_PROFILE_TABLE WHERE PROFILEID > 0;
     ```

   - Issue a BIND or REBIND command and specify EXPLAIN(YES).

b. Query the EXPLAIN data on the test subsystem to compare values to those values that you captured from the production system. For example, you might issue the following series of statements:

   ```
   SELECT CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),1,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),1,1)) END
         * POWER (16, 7)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),2,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,69,1)),2,1)) END
    * POWER (16, 6)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),1,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),1,1)) END
    * POWER (16, 5)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),2,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,70,1)),2,1)) END
    * POWER (16, 4)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),1,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
      ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),1,1)) END
    * POWER (16, 3)
    + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),2,1)
        WHEN 'A' THEN 10
        WHEN 'B' THEN 11
        WHEN 'C' THEN 12
        WHEN 'D' THEN 13
        WHEN 'E' THEN 14
        WHEN 'F' THEN 15
   ```

```
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,71,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,72,1)),2,1)) END
 * POWER (16, 0) AS CPUSPEED_EXP,
FROM PLAN_TABLE
WHERE QUERYNO = 1;

SELECT CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),1,1)) END
      * POWER (16, 3)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,25,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,26,1)),2,1)) END
 * POWER (16, 0) AS NUMCP_EXP,FROM PLAN_TABLE
WHERE QUERYNO = 1;

SELECT CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),1,1)) END
      * POWER (16, 7)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,13,1)),2,1)) END
 * POWER (16, 6)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),1,1)) END
 * POWER (16, 5)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,14,1)),2,1)) END
 * POWER (16, 4)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),1,1)) END
 * POWER (16, 3)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,15,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,16,1)),2,1)) END
 * POWER (16, 0) AS RIDPOOL_EXP,
ATTRIBUTE2 AS RIDPOOL_PROFILE
```

```
FROM PLAN_TABLE, SYSIBM.DSN_PROFILE_ATTRIBUTES
WHERE KEYWORDS= 'MAX_RIDBLOCKS'
AND QUERYNO = 1;


SELECT CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),1,1)) END
      * POWER (16, 7)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,9,1)),2,1)) END
 * POWER (16, 6)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),1,1)) END
 * POWER (16, 5)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,10,1)),2,1)) END
 * POWER (16, 4)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),1,1)) END
 * POWER (16, 3)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,11,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,12,1)),2,1)) END
 * POWER (16, 0) AS SORTPOOL_EXP,
ATTRIBUTE2 AS SORTPOOL_PROFILE
FROM PLAN_TABLE, SYSIBM.DSN_PROFILE_ATTRIBUTES
WHERE KEYWORDS= 'SORT_POOL_SIZE'
AND QUERYNO = 1;

SELECT QUERYNO,TBL.CREATOR,TBL.NAME,TS.NAME,
BPOOL,ATTRIBUTE2 AS BP_PROFILE,
CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,1,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,1,1)),1,1)) END
      * POWER (16, 7)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,1,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,1,1)),2,1)) END
 * POWER (16, 6)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,2,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,2,1)),1,1)) END
 * POWER (16, 5)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,2,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,2,1)),2,1)) END
 * POWER (16, 4)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,3,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,3,1)),1,1)) END
 * POWER (16, 3)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,3,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,3,1)),2,1)) END
 * POWER (16, 2)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,4,1)),1,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,4,1)),1,1)) END
 * POWER (16, 1)
 + CASE SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,4,1)),2,1)
     WHEN 'A' THEN 10
     WHEN 'B' THEN 11
     WHEN 'C' THEN 12
     WHEN 'D' THEN 13
     WHEN 'E' THEN 14
     WHEN 'F' THEN 15
   ELSE  INTEGER(SUBSTR(HEX(SUBSTR(IBM_SERVICE_DATA,4,1)),2,1)) END
 * POWER (16, 0) AS BP_EXP
FROM SYSIBM.SYSTABLESPACE TS, SYSIBM.SYSTABLES TBL,
SYSIBM.DSN_PROFILE_ATTRIBUTES,PLAN_TABLE
WHERE KEYWORDS = BPOOL
AND TS.NAME = TSNAME
AND TBL.CREATOR = PLAN_TABLE.CREATOR
AND TBL.NAME = PLAN_TABLE.TNAME
AND QUERYNO = 1;

SELECT QUERYNO,REASON FROM DSN_STATEMNT_TABLE
WHERE QUERYNO = 1;
```

The following tables show example results from the preceding statements:

CPUSPEED_EXP
380

NUMCP_EXP
1

RIDPOOL_EXP	RIDPOOL_PROFILE
300	300

SORTPOOL_EXP	SORTPOOL_PROFILE
307200	307200

BPOOL	BP_PROFILE	BP_EXP
BP0	25000	25000

Chapter 49. Modeling your production system statistics in a test subsystem

You can improve access path testing by updating the catalog statistics on your test system to be the same as your production system.

About this task

When you bind applications on the test system that uses production statistics, access paths are more likely to match for queries bound in each environment. However, corresponding access paths from test and production environments might differ for the following possible reasons:

- The resources available in the test subsystem, such as the type and number of processors, or RID pool, sort pool, and buffer pool settings do not match those of the production environment. However, you can create profiles and set subsystem parameters on your test subsystem to better model your production environment.
- Data in SYSIBM.SYSCOLDIST is mismatched. (This mismatch occurs only if some of the previously mentioned steps mentioned are not followed exactly).
- The service levels are different.
- The values of optimization subsystem parameters, such as STARJOIN, NPGTHRSH, and PARAMDEG (MAX DEGREE on installation panel DSNTIP8) are different. You can prevent this problem by setting these parameters in your test subsystem to match your production environment.
- The use of techniques such as optimization hints and volatile tables are different.

If your production system is accessible from your test system, you can use DB2 PM EXPLAIN on your test system to request EXPLAIN information from your production system. This request can reduce the need to simulate a production system by updating the catalog.

You can also use IBM Data Studio, IBM Data Server Manager, or DB2 Query Workload Tuner for z/OS to compare access plan graphs and capture information about your DB2 environment.

Important: Consider the following information when using SPUFI:

- If you use SPUFI to execute the following example SQL statements, you might need to increase the default maximum character column width to avoid truncation.
- Asterisks (*) are used in the examples to avoid having the semicolon interpreted as the end of the SQL statement. Edit the result to change the asterisk to a semicolon.

Procedure

To model production statistics in your test environment:

1. Run RUNSTATS on your production tables to get current statistics for access path selection.
2. Retrieve the production statistics and use them to build SQL statements to update the catalog of the test system. You can use queries similar to the

following queries to build those statements. To successfully model your production system, the table definitions must be the same on both the test and production systems. For example, they must have the same creator, name, indexes, number of partitions, and so on.

a.

PSPI

Use the following statements to update SYSTABLESPACE, SYSTABLES, SYSINDEXES, and SYSCOLUMNS:

```
SELECT DISTINCT 'UPDATE SYSIBM.SYSTABLESPACE SET NACTIVEF='
CONCAT STRIP(CHAR(NACTIVEF))
CONCAT',NACTIVE='CONCAT STRIP(CHAR(NACTIVE))
CONCAT ' WHERE NAME=''' CONCAT TS.NAME
CONCAT ''' AND DBNAME ='''CONCAT TS.DBNAME CONCAT'''*'
FROM SYSIBM.SYSTABLESPACE TS, SYSIBM.SYSTABLES TBL
WHERE TS.NAME = TSNAME
      AND TBL.CREATOR IN (table creator_list)
      AND TBL.NAME IN (table_list)
      AND (NACTIVEF >=0 OR NACTIVE  >=0);

SELECT 'UPDATE SYSIBM.SYSTABLES SET CARDF='
CONCAT  STRIP(CHAR(CARDF))
CONCAT',NPAGES='CONCAT STRIP(CHAR(NPAGES))
CONCAT',NPAGESF='CONCAT STRIP(CHAR(NPAGESF))
CONCAT',PCTROWCOMP='CONCAT STRIP(CHAR(PCTROWCOMP))
CONCAT ' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT'''*'
FROM SYSIBM.SYSTABLES WHERE
CREATOR IN (creator_list)
        AND NAME IN (table_list)
        AND CARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSINDEXES SET FIRSTKEYCARDF='
CONCAT  STRIP(CHAR(FIRSTKEYCARDF))
CONCAT ',FULLKEYCARDF='CONCAT STRIP(CHAR(FULLKEYCARDF))
CONCAT',NLEAF='CONCAT STRIP(CHAR(NLEAF))
CONCAT',NLEVELS='CONCAT STRIP(CHAR(NLEVELS))
CONCAT',CLUSTERRATIO='CONCAT STRIP(CHAR(CLUSTERRATIO))
CONCAT',CLUSTERRATIOF='CONCAT STRIP(CHAR(CLUSTERRATIOF))
CONCAT',DATAREPEATFACTORF='CONCAT STRIP(CHAR(DATAREPEATFACTORF))
CONCAT' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT'''*'
FROM SYSIBM.SYSINDEXES
WHERE TBCREATOR IN (creator_list)
      AND TBNAME IN (table_list)
      AND FULLKEYCARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSCOLUMNS SET COLCARDF='
CONCAT STRIP(CHAR(COLCARDF))
CONCAT',HIGH2KEY= X''' CONCAT HEX(HIGH2KEY)
CONCAT''',LOW2KEY= X''' CONCAT HEX(LOW2KEY)
CONCAT''' WHERE TBNAME=''' CONCAT TBNAME CONCAT  ''' AND COLNO='
CONCAT STRIP(CHAR(COLNO))
CONCAT ' AND TBCREATOR =''' CONCAT TBCREATOR CONCAT'''*'
FROM SYSIBM.SYSCOLUMNS
WHERE TBCREATOR IN (creator_list)
      AND TBNAME IN (table_list)
      AND COLCARDF >= 0;
```

PSPI

b.

GUPI

Delete statistics from SYSTABSTATS on the test subsystem for the specified tables by using the following statement:

```
DELETE FROM (TEST_SUBSYSTEM).SYSTABSTATS
   WHERE OWNER IN (creator_list)
   AND   NAME IN (table_list);
```

GUPI

c.

PSPI

Use INSERT statements to repopulate SYSTABSTATS with production statistics that are generated from the following statement:

```
SELECT 'INSERT INTO SYSIBM.SYSTABSTATS'
CONCAT '(CARD,NPAGES,PCTPAGES,NACTIVE,PCTROWCOMP'
CONCAT ',STATSTIME,IBMREQD,DBNAME,TSNAME,PARTITION'
CONCAT ',OWNER,NAME,CARDF) VALUES('
CONCAT  STRIP(CHAR(CARD))             CONCAT ' ,'
CONCAT  STRIP(CHAR(NPAGES))           CONCAT ' ,'
CONCAT  STRIP(CHAR(PCTPAGES))         CONCAT ' ,'
CONCAT  STRIP(CHAR(NACTIVE))          CONCAT ' ,'
CONCAT  STRIP(CHAR(PCTROWCOMP))       CONCAT ' ,'
CONCAT  '''' CONCAT CHAR(STATSTIME)   CONCAT ''' ,'
CONCAT  '''' CONCAT IBMREQD           CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(DBNAME)     CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(TSNAME)     CONCAT ''' ,'
CONCAT  STRIP(CHAR(PARTITION))        CONCAT ' ,'
CONCAT  '''' CONCAT STRIP(OWNER)      CONCAT ''' ,'
CONCAT '''' CONCAT STRIP(NAME)        CONCAT ''' ,'
CONCAT  STRIP(CHAR(CARDF))            CONCAT  ')*'
FROM  SYSIBM.SYSTABSTATS
      WHERE OWNER IN (creator_list)
      AND   NAME IN (table_list);
```

PSPI

d.

GUPI

Delete statistics from SYSCOLDIST on the test subsystem for the specified tables by using the following statement:

```
DELETE FROM (TEST_SUBSYSTEM).SYSCOLDIST
   WHERE TBOWNER IN (creator_list)
   AND   TBNAME IN (table_list);
```

GUPI

e.

PSPI

Use INSERT statements to repopulate SYSCOLDIST with production statistics that are generated from the following statement:

```
SELECT 'INSERT INTO SYSIBM.SYSCOLDIST '
CONCAT '(FREQUENCY,STATSTIME,IBMREQD,TBOWNER'
CONCAT ',TBNAME,NAME,COLVALUE,TYPE,CARDF,COLGROUPCOLNO'
CONCAT ',NUMCOLUMNS,FREQUENCYF,QUANTILENO,LOWVALUE'
CONCAT ',HIGHVALUE) VALUES('
CONCAT  STRIP(CHAR(FREQUENCY))            CONCAT ' ,'
CONCAT  '''' CONCAT CHAR(STATSTIME)       CONCAT ''' ,'
CONCAT  '''' CONCAT IBMREQD               CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(TBOWNER)        CONCAT ''' ,'
CONCAT  '''' CONCAT STRIP(TBNAME)         CONCAT ''','
CONCAT  '''' CONCAT STRIP(NAME)           CONCAT ''' ,'
CONCAT  'X''' CONCAT STRIP(HEX(COLVALUE)) CONCAT ''' ,'
CONCAT  '''' CONCAT TYPE                  CONCAT ''' ,'
CONCAT  STRIP(CHAR(CARDF))                CONCAT ' ,'
CONCAT  'X'''CONCAT STRIP(HEX(COLGROUPCOLNO)) CONCAT ''' ,'
CONCAT  CHAR(NUMCOLUMNS)                  CONCAT ' ,'
CONCAT  STRIP(CHAR(FREQUENCYF))           CONCAT ','
CONCAT  CHAR(QUANTILENO)                  CONCAT ','
CONCAT  'X''' CONCAT STRIP(HEX(LOWVALUE)) CONCAT ''','
CONCAT  'X''' CONCAT STRIP(HEX(HIGHVALUE)) CONCAT ''')*'
FROM  SYSIBM.SYSCOLDIST
      WHERE TBOWNER IN (creator_list)
      AND   TBNAME IN (table_list);
```

PSPI

Related concepts:

Interpreting data access by using EXPLAIN

Investigating SQL performance by using EXPLAIN

Related tasks:

Modeling a production environment on a test subsystem

Part 11. Enabling DB2 for IBM DB2 Analytics Accelerator for z/OS

In the DB2 for z/OS environment, an *accelerator* is a solution that can run queries on behalf of DB2.

In addition to any other accelerators that might be available in the marketplace, IBM offers a high-performance accelerator appliance: IBM DB2 Analytics Accelerator for z/OS. The information in the following topics assumes that you use IBM DB2 Analytics Accelerator for z/OS, or an equivalent accelerator.

An *accelerator server* is a specific instance of an accelerator. For certain types of queries, such as business intelligence (BI) queries, processing on an accelerator server can be faster than processing on DB2.

Before you begin

For applications that run on a remote client or driver, the level of that client or driver must support DB2 9 in new-function mode or later. The following types of clients and drivers are the minimum levels that are supported. Later versions are required to support some functions:

- Non-Java clients or drivers that are shipped with DB2 for Linux, UNIX, and Windows at the Version 9.1 GA level
- IBM Data Server Driver for JDBC and SQLJ Version 3.1.57

For more information about the hardware and software requirements for installing IBM IBM DB2 Analytics Accelerator for z/OS, see Prerequisites and Maintenance forIBM DB2 Analytics Accelerator for z/OS Version 4.1.

Procedure

To enable DB2 to work with an accelerator:

1. Install and configure an accelerator server. For information about the installation and configuration requirements for IBM IBM DB2 Analytics Accelerator for z/OS, see the IBM DB2 Analytics Accelerator for z/OS documentation. The product documentation for the accelerator also includes information about installing and setting up the stored procedures that enable the accelerator to work with DB2.
2. Create a database, a table space, tables, and indexes that are used to support the acceleration of queries.
3. Enable DB2 to send queries to the accelerator server by setting subsystem parameters, special registers, and bind options.
   - The ACCEL subsystem parameter controls whether a DB2 subsystem can use accelerator servers.
   - The QUERY_ACCELERATION subsystem parameter and the CURRENT QUERY ACCELERATION special register control whether DB2 considers dynamic queries for acceleration and how DB2 behaves when acceleration fails.
   - The QUERY_ACCEL_OPTIONS subsystem parameter specifies other types of SQL queries that are eligible for acceleration.

- The QUERYACCELERATION bind option controls whether a static SQL query is bound for acceleration, and if so, with what behavior.
- The GET_ACCEL_ARCHIVE subsystem parameter and the CURRENT GET_ACCEL_ARCHIVE special register control whether a dynamic SQL query that references a table that is archived on an accelerator server uses the archived data.
- The GETACCELARCHIVE bind option specifies whether a static SQL query that is bound for acceleration retrieves archived data on the accelerator, instead of active data.

The following table shows the settings that enable the sending of queries to accelerator servers.

Subsystem parameter, special register, or bind option	Valid settings
ACCEL	COMMAND or AUTO
QUERY_ACCELERATION (if CURRENT QUERY ACCELERATION is not set)	ENABLE, ENABLE_WITH_FAILBACK, ELIGIBLE, or ALL
CURRENT QUERY ACCELERATION	ENABLE, ENABLE WITH FAILBACK, ELIGIBLE, or ALL
QUERYACCELERATION	NONE, ENABLE, ENABLEWITHFAILBACK, ELIGIBLE, or ALL
QUERY_ACCEL_OPTIONS	NONE, 1, 2, 3, or 4
GET_ACCEL_ARCHIVE (if CURRENT GET_ACCEL_ARCHIVE is not set)	NO or YES
CURRENT GET_ACCEL_ARCHIVE	NO or YES
GETACCELARCHIVE	NO or YES

4. Identify the accelerator servers that are available to DB2 by issuing the -START ACCEL command. On successful completion of the -START ACCEL command, queries for the specified accelerator servers can begin to run. Alternatively, if you specify the ACCEL subsystem parameter as AUTO, accelerator servers are automatically enabled and started when the DB2 subsystem is started.

   Also, you can use either the DB2 Administration Tool for z/OS or IBM DB2 Analytics Accelerator Studio to enable and start accelerator servers. Both of these products enable accelerator servers by invoking the -START ACCEL command on a DB2 subsystem.

What to do next

Tip: The DB2 Administration Tool for z/OS is one product that you can use to customize parameters and manage accelerator servers for IBM DB2 Analytics Accelerator for z/OS. With the DB2 Administration Tool for z/OS, you can start, stop, add, and delete accelerator servers. You can also use the DB2 Administration Tool for z/OS to populate, maintain, and control accelerated tables. For more information, see Using IBM DB2 Analytics Accelerator for z/OS (DB2 Administration Tool for z/OS).

Related concepts:

How DB2 determines whether to accelerate eligible queries

Related tasks:

Monitoring the use of accelerators for DB2 for z/OS queries

Related reference:

Reliability and Performance with DB2Version 4.1 (IBM Redbooks)

Hybrid Analytics Solution using IBM DB2 Analytics Accelerator for z/OS V3.1 (IBM Redbooks)

Optimizing DB2 Queries with IBM DB2 Analytics Accelerator for z/OS (IBM Redbooks)

Support Portal: IBM DB2 Analytics Accelerator for z/OS

Reference information for working with accelerators

Special registers (DB2 SQL)

-START ACCEL (DB2) (DB2 Commands)

Chapter 50. How DB2 determines whether to accelerate eligible queries

Only eligible queries are candidates to be passed to an accelerator server. However, even if a query is eligible for acceleration, DB2 might not pass the query to an accelerator server.

PSPI

Here are the criteria that make a query eligible to be passed to an accelerator server:

- The accelerator must support all of the SQL functions in the query.
- The tables that the query references must be in the accelerator server.
- Each table that the query references must have a row in the SYSACCEL.SYSACCELERATEDTABLES table.
- The query is dynamic or static and is defined as read-only.
- The query is a SELECT statement, an INSERT with SELECT statement, or a local SELECT INTO statement.
- The static query is not a remote SELECT INTO statement and is not part of an expression of the SET host-variable assignment statement.
- The cursor for the query is not a scrollable or rowset cursor.
- All of the other conditions are true, and DB2 determines that the query should be accelerated.

After a query is determined to be eligible, DB2 bases the decision to pass a query to the accelerator server on the following factors:

- DB2 determines that it can run the query more quickly than the accelerator. In general, for a short-running query, DB2 requires less time to run a query than it would take to send the query to the accelerator server and receive the data from the accelerator server. This criterion is used only in the following cases:
  - If the QUERY_ACCELERATION subsystem parameter is set to ENABLE or ENABLE_WITH_FAILBACK
  - If the CURRENT QUERY ACCELERATION special register overrides the QUERY_ACCELERATION value with a setting of ENABLE or ENABLE WITH FAILBACK
  - If the DB2 package is bound with the QUERYACCELERATION bind option value of ENABLE or ENABLEWITHFAILBACK
- The isolation level for the query is not supported by the accelerator server.
- DB2 cannot guarantee that the results that are generated by the accelerator server are the same as the results that are generated by DB2.

An accelerator server is qualified to process a query if it is active and contains all of the tables that are referenced in the query. If more than one accelerator server is qualified to process a query, DB2 either sends the query to the first qualified accelerator server, or if the accelerator supports workload balancing, DB2 sends the query to the accelerator server that has greater capacity and lower utilization.

PSPI

Related concepts:

What happens when acceleration fails

Related tasks:

Enabling DB2 for IBM DB2 Analytics Accelerator for z/OS

Related reference:

Optimizing DB2 Queries with IBM DB2 Analytics Accelerator for z/OS (IBM Redbooks)

CURRENT QUERY ACCELERATION (DB2 SQL)

SYSACCEL.SYSACCELERATEDTABLES table (DB2 SQL)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Determining whether queries can benefit from acceleration

To evaluate the potential savings for both accumulated elapsed time and processor time, you can enable the modeling of query workloads for plans that run on an accelerator server.

About this task

Results are recorded in the accelerator fields of accounting trace IFCID 003. Only queries that DB2 deems eligible to run on an accelerator server are included in IFCID 003.

Procedure

To determine whether queries can benefit from acceleration:

Enable the modeling of query workloads for plans that run on an accelerator server by setting the following subsystem parameters and special registers:

- Set the ACCEL_MODEL subsystem parameter to YES
- Set the QUERY ACCELRATION subsystem parameter value to NONE, ENABLE, ENABLE WITH FAILBACK or ELIGIBLE.
- Set the CURRENT GET_ACCEL_ARCHIVE subsystem parameter to NO.
- Set the CURRENT GET_ACCEL_ARCHIVE special register to NO

If other values are specified, DB2 tries to accelerate queries instead of doing accelerator modeling.

Related concepts:

How DB2 determines whether to accelerate eligible queries

Related tasks:

Monitoring the use of accelerators for DB2 for z/OS queries

Related reference:

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Reference information for working with accelerators

What happens when acceleration fails

If a failure occurs while a query is running on an accelerator server, DB2 runs the query under certain conditions.

DB2 runs the query in the following cases:

- The setting for the QUERY_ACCELERATION subsystem parameter is 2 (ENABLE_WITH_FAILBACK), and the CURRENT QUERY ACCELERATION special register is not set.
- The setting for CURRENT QUERY ACCELERATION is ENABLE WITH FAILBACK.

Related concepts:

How DB2 determines whether to accelerate eligible queries

Related reference:

SET CURRENT QUERY ACCELERATION (DB2 SQL)

CURRENT QUERY ACCELERATION (DB2 SQL)

Subsystem parameters that are not on installation panels (DB2 Installation and Migration)

Chapter 51. Monitoring the use of accelerators for DB2 for z/OS queries

You can use DB2 traces to monitor how DB2 uses query accelerators.

Procedure

To monitor the use of query accelerators:

- Use the accounting trace. The DB2 accounting class 1 trace records provide information about how often accelerators were used and how often accelerators failed.
- Use the statistics trace. The DB2 statistics class 1 trace provides:
  - The states of the accelerators that are in use
  - The amount of processing time that is spent in accelerators
  - Counts of the amounts of sent and received information
  - Counts of the number of times that queries were successfully and unsuccessfully processed by accelerators

Related concepts:

Accounting trace

Statistics trace

How DB2 determines whether to accelerate eligible queries

Related tasks:

Enabling DB2 for IBM DB2 Analytics Accelerator for z/OS

Related reference:

Support Portal: IBM DB2 Analytics Accelerator for z/OS

Chapter 52. Using an alias for an accelerator

When you need a DB2 application to run on an accelerator in a different subsystem, you can use logical names, called aliases. Using aliases can help make the application portable so that you can run the application, without modification, on an accelerator in another subsystem.

For example, you might have an application that runs initially on an accelerator in your test subsystem, and then later you want to run it in a production subsystem. Similarly, you might run an application in one production subsystem, but you also need to run the application on accelerators in multiple other production subsystems. In both cases, using an alias is an efficient way to make the application portable across accelerators in different subsystems.

Before you begin

Ensure that the physical name for the accelerator in a specific subsystem is already defined.

About this task

Defining an accelerator alias enables you to map a logical name (alias) to a physical accelerator name that is used in a specific subsystem. A DB2 application can then use that alias, which points to an accelerator in that subsystem. You can also set up multiple aliases to point to the same accelerator, if needed.

Restrictions:

- You cannot define a chain of aliases for accelerators. That is, you cannot have one alias point to another alias, which points to yet another alias.
- DB2 performs only a single lookup for the mapping of a logical name (location alias) to a physical accelerator name in that same subsystem for each CREATE TABLE statement.

Procedure

To create an alias for an accelerator:

Create special rows in the SYSIBM.LOCATIONS communications database (CDB) table of one subsystem. For example:

GUPI

```
INSERT INTO SYSIBM.LOCATIONS (LOCATION, LINKNAME, DBALIAS)
VALUES ('logical_system_accel_name', 'DSNACCELERATORALIAS',
'physical_system_accel_name')
```

GUPI

In this example:

logical_system_accel_name
Is the name of an accelerator in another DB2 subsystem, typically a production subsystem, where the accelerator is defined. A logical name is used only when you issue a CREATE TABLE...IN ACCELERATOR *accel_name* statement.

For example, if the name of the accelerator on your production system is PACCEL, and the name of the accelerator on your test system is TACCEL, create an alias in SYSIBM.LOCATIONS on your test system in which LOCATION = 'PACCEL' and DBALIAS= 'TACCEL'. By doing so, when you port an application that uses accelerator-only tables from your test environment to your production environment, the DDL will not need to be changed.

physical_system_accel_name
Is the real name of an accelerator or accelerators in this DB2 subsystem.

Results

When DB2 processes the CREATE TABLE...IN ACCELERATOR *logical_system_accel_name* clause, DB2 first looks for the accelerator named *logical_system_accel_name* in that same subsystem. If an accelerator with that name does not exist in that subsystem, DB2 checks the SYSIBM.LOCATION table for the corresponding DSNACCELERATORALIAS value, which identifies the physical accelerator name in the that subsystem.

Chapter 53. Types of accelerator tables

There are three types of accelerator tables.

Accelerator-only table

An accelerator-only table is automatically created in the accelerator when the CREATE TABLE statement is issued on DB2 with the IN ACCELERATOR clause. The table and column definitions of the accelerator-only table are reflected in the DB2 catalog with a D in the TYPE column of SYSIBM.SYSTABLES.

You can specify an alias (logical name) when defining an accelerator. For more information, see Using an alias for an accelerator.

Any queries that reference an accelerator-only table must be executed in the accelerator and will be accelerated. If the query is not eligible for query acceleration, an error is returned.

A data change statement for an accelerator-only table must be executed in the accelerator. If the data change statement type is not supported by the accelerator, or if the statement contains any expression that is not supported by the accelerator, an error is returned.

When validate run behavior is in effect, a static query or data change statement that references an accelerator-only table is incrementally bound at run time. A static query or data change statement that references an accelerator-only table is eligible for acceleration during an incremental bind only if the QUERYACCELERATION bind option that is in effect is ENABLE or ELIGIBLE. When a static query or data change statement that references an accelerator-only table is issued and the QUERYACCELERATION bind option that is in effect is ALL, an error is returned.

Running queries and data change statements on the accelerator can significantly speed up SQL statements, such as INSERT from SELECT statements. This is not possible with the other two types of accelerator tables. Accelerator-only tables are used by statistics and analytics tools, which can quickly gather all the data that is required for reports. Because the data in these tables can be modified so quickly, they are also ideal for data-preparation tasks that must be completed before the data can be used for predictive modeling.

Accelerator-shadow table

An accelerator-shadow table exists both in DB2 and in the accelerator. The table in the accelerator contains all or a subset of the columns in the DB2 table. After the table is defined on the accelerator, you can load data into the table on the accelerator by copying data from the original DB2 table to the corresponding table on the accelerator. After the data is loaded on the accelerator, you can enable query acceleration for this table by using one of the following methods: QUERY_ACCELERATION subsystem parameter, CURRENT QUERY ACCELERATION special register, QUERYACCELERATION bind option, or connection properties (JDBC and ODBC) and profile tables.

Accelerator-archived table

An accelerator-archived table is a table on the accelerator for which partitions of a DB2 table, or the entire table, are archived on the accelerator. When the data is archived on the accelerator, the original DB2 data is deleted. The table space for a DB2 table that is associated with an archived accelerator table is set to a permanent read-only state so that the original DB2 table can no longer be changed. An image copy of the data is also created as part of the archive, to enable recovery of the data in an emergency situation. The recovery function in IBM DB2 Analytics Accelerator for z/OS uses the image copies to restore the data in the original DB2 table.

Accelerator-archived tables are used primarily for historical data that is no longer actively used or maintained. The archive saves storage space on IBM Z. Normally, the archived data is not included in an accelerated query. However, you can specify that archived data is to be included in an accelerated query by using one of the following methods: QUERY_ACCELERATION subsystem parameter, CURRENT QUERY ACCELERATION special register, QUERYACCELERATION bind option, and connection properties (JDBC and ODBC) and profile tables.

For certain scenarios, query results for the accelerator tables might differ from results that are executed in DB2. For more information, see the information about restrictions in IBM DB2 Analytics Accelerator for z/OStables.

Related information:

Restrictions of accelerator-only tables

Chapter 54. Reference information for working with accelerators

DB2 provides tables, commands, special registers, and SQL statements that support the interactions between DB2 and accelerator servers.

Tables that support query acceleration (DB2 SQL)

-START ACCEL (DB2) (DB2 Commands)

-DISPLAY ACCEL (DB2) (DB2 Commands)

-STOP ACCEL (DB2) (DB2 Commands)

CURRENT GET_ACCEL_ARCHIVE (DB2 SQL)

CURRENT QUERY ACCELERATION (DB2 SQL)

SET CURRENT GET_ACCEL_ARCHIVE (DB2 SQL)

SET CURRENT QUERY ACCELERATION (DB2 SQL)

Part 12. Appendixes

Appendix A. DB2-supplied stored procedures for managing performance

DB2 provides stored procedures that you can call in your application programs to perform performance management functions. Typically, these procedures are created during installation or migration.

Appendix B. DB2-supplied user tables

DB2-supplied user tables are tables that you or certain tools might create on your DB2 for z/OS subsystem.

Supplied user tables enable you to capture information about data access by using the EXPLAIN function, to monitor query and system performance with monitor profiles, and to limit the use of resources by particular applications or middleware servers by using the resource limit facility. Instances of these tables also enable optimization tools to capture, analyze, and store information about query performance.

EXPLAIN tables

EXPLAIN tables contain information about SQL statements and functions that run on DB2 for z/OS.

You can create and maintain a set of *EXPLAIN tables* to capture and analyze information about the performance of SQL statements and functions that run on DB2 for z/OS. Each row in an EXPLAIN table describes some aspect of a step in the execution of a query or subquery in an explainable statement. The column values for the row identify, among other things, the query or subquery, the tables and other objects involved, the methods used to carry out each step, and cost information about those methods. DB2 creates EXPLAIN output and populates EXPLAIN tables in the following situations:

- When an EXPLAIN statement is executed.
- At BIND or REBIND with the EXPLAIN(YES) or (ONLY) bind options. Rows are added for every explainable statement in the plan or package being bound. For a plan, these do not include statements in the packages that can be used with the plan. For either a package or plan, they do not include explainable statements within EXPLAIN statements nor do they include explainable statements that refer to declared temporary tables, which are incrementally bound at run time.
- When an explainable dynamic statement is executed and the value of the CURRENT EXPLAIN MODE special register is set to YES or EXPLAIN.
- When the DSNAEXP stored procedure executes successfully. The DSNAEXP stored procedure is deprecated.

Important: It is best to convert EXPLAIN tables to DB2 10 format during migration, or soon after migration. In DB2 10, the EXPLAIN function supports tables that have only the DB2 10, DB2 9, or Version 8 formats. However, DB2 9 format and Version 8 format EXPLAIN tables are deprecated. If you invoke EXPLAIN and DB2 9 or Version 8 tables are used, DB2 issues SQL code +20520. If tables of an unsupported format are found, DB2 issues SQL code -20008 and the EXPLAIN operation fails.

Important: If the EXPLAIN tables have any format older than the DB2 Version 8 format, or are encoded in EBCDIC, DB2 returns an error for any operation that tries inserts rows in the EXPLAIN tables.

Related tasks:

➪ Migration step 24: Convert EXPLAIN tables to the current format and encoding type (DB2 Installation and Migration)

Related reference:

BIND and REBIND options for packages and plans (DB2 Commands)

Interpreting data access by using EXPLAIN

Capturing access path information in EXPLAIN tables

PLAN_TABLE

The plan table, PLAN_TABLE, contains information about access paths that is collected from the results of EXPLAIN statements.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Optional PLAN_TABLE formats

A PLAN_TABLE instance can have a format with fewer columns than those shown in the sample CREATE TABLE statement. However instances of PLAN_TABLE must have one of the following formats:

DB2 10 format
All columns shown in the sample CREATE TABLE statement, up to and including the MERGN column (COLCOUNT=64).

DB2 9 format
All columns shown in the sample CREATE TABLE statement, to and including the PARENT_PLANNO column (COLCOUNT=59). This format is deprecated. For information about converting tables in this format to the

current format, see Migration step 24: Convert EXPLAIN tables to the current format and encoding type (DB2 Installation and Migration).

DB2 Version 8 format
All columns shown in the sample CREATE TABLE statement, up to and including the STMTTOKEN column (COLCOUNT=58). This format is deprecated. For information about converting tables in this format to the current format, see Converting EXPLAIN tables for migration from DB2 Version 8 (DB2 Installation and Migration).

Important: If the EXPLAIN tables have any format older than the DB2 Version 8 format, or are encoded in EBCDIC, DB2 returns an error for any operation that tries inserts rows in the EXPLAIN tables.

Column descriptions

Your subsystem or data sharing group can contain more than one of these tables, including a table with the qualifier SYSIBM, a table with the qualifier DB2OSCA, and additional tables that are qualified by user IDs.

The following table shows the descriptions of the columns in PLAN_TABLE.

Table 184. Descriptions of columns in PLAN_TABLE

Column name	Data Type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
PLANNO	SMALLINT NOT NULL	The number of the step in which the query that is indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.
METHOD	SMALLINT NOT NULL	A number that indicates the join method that is used for the step: **0** The table in this step is the first table that is accessed, a continuation of a previous table that was accessed, or a table that is not used. **1** A nested loop join is used. For each row of the current composite table, matching rows of a new table are found and joined. **2** A merge scan join is used. The current composite table and the new table are scanned in the order of the join columns, and matching rows are joined. **3** Sorts are needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, INTERSECT, EXCEPT, a quantified predicate, or an IN predicate. This step does not access a new table. **4** A hybrid join was used. The current composite table is scanned in the order of the join-column rows of the new table. The new table is accessed using list prefetch.
CREATOR	VARCHAR(128) NOT NULL	The creator of the new table that is accessed in this step, blank if METHOD is 3.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
TNAME	VARCHAR(128) NOT NULL	The name of one of the following objects: • Table • Materialized query table • Created or declared temporary table • Materialized view • Materialized table expression • Any of the following object names that identify intermediate results: **'DSNWFQB(*qblockno*)'** The intermediate result of a UNION ALL, INTERSECT ALL, EXCEPT ALL, or an outer join that is materialized. If a view is merged, the name of the view does not appear. **'DSN_DIM_TBLX(*qblockno*)'** The work file of a star join dimension table. The value is blank if METHOD is 3.
TABNO	SMALLINT NOT NULL	Values are for IBM use only.
ACCESSTYPE[1]	CHAR(2) NOT NULL	The method of accessing the new table.[4]
MATCHCOLS	SMALLINT NOT NULL	For ACCESSTYPE I, IN, I1, N, NR, MX, or DX, the number of index keys that are used in an index scan; otherwise, 0.
ACCESSCREATOR	VARCHAR(128) NOT NULL	For ACCESSTYPE I, I1, N, NR, MX, or DX, the creator of the index; otherwise, blank.
ACCESSNAME	VARCHAR(128) NOT NULL	For ACCESSTYPE I, I1, H, MH, N, NR, MX, or DX, the name of the index; for ACCESSTYPE P, DSNPJW(*mixopseqno*) is the starting pair-wise join leg in MIXOPSEQ; otherwise, blank.
INDEXONLY	CHAR(1) NOT NULL	Indication of whether access to an index alone is enough to perform the step, or Indication of whether data too must be accessed. **Y** Yes **N** No
SORTN_UNIQ	CHAR(1) NOT NULL	Indication of whether the new table is sorted to remove duplicate rows. **Y** Yes **N** No
SORTN_JOIN	CHAR(1) NOT NULL	Indication of whether the new table is sorted for join method 2 or 4. **Y** Yes **N** No
SORTN_ORDERBY	CHAR(1) NOT NULL	Indication of whether the new table is sorted for ORDER BY. **Y** Yes **N** No

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
SORTN_GROUPBY	CHAR(1) NOT NULL	Indication of whether the new table is sorted for GROUP BY. **Y** Yes **N** No
SORTC_UNIQ	CHAR(1) NOT NULL	Indication of whether the composite table is sorted to remove duplicate rows. **Y** Yes **N** No
SORTC_JOIN	CHAR(1) NOT NULL	Indication of whether the composite table is sorted for join method 1, 2 or 4. **Y** Yes **N** No
SORTC_ORDERBY	CHAR(1) NOT NULL	Indication of whether the composite table is sorted for an ORDER BY clause or a quantified predicate. **Y** Yes **N** No
SORTC_GROUPBY	CHAR(1) NOT NULL	Indication of whether the composite table is sorted for a GROUP BY clause. **Y** Yes **N** No

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
TSLOCKMODE	CHAR(3) NOT NULL	An indication of the mode of lock that is acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time, the values are: **IS** Intent share lock **IX** Intent exclusive lock **S** Share lock **U** Update lock **X** Exclusive lock **SIX** Share with intent exclusive lock **N** UR isolation; no lock If the isolation level cannot be determined at bind time, the lock mode is determined by the isolation level at run time is shown by the following values. **NS** For UR isolation, no lock; for CS, RS, or RR, an S lock. **NIS** For UR isolation, no lock; for CS, RS, or RR, an IS lock. **NSS** For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock. **SS** For UR, CS, or RS isolation, an IS lock; for RR, an S lock. The data in this column is right justified. For example, IX appears as a blank, followed by I, followed by X. If the column contains a blank, then no lock is acquired. If the access method in the ACCESSTYPE column is DX, DI, or DU, no latches are acquired on the XML index page and no lock is acquired on the new base table data page or row, nor on the XML table and the corresponding table spaces. The value of TSLOCKMODE is a blank in this case.
TIMESTAMP	CHAR(16) NOT NULL	This column is deprecated. Use EXPLAIN_TIME instead.
REMARKS	VARCHAR(762) NOT NULL	A field into which you can insert any character string of 762 or fewer characters. DB2 inserts a value into this column in certain situations. [6] ,
PREFETCH	CHAR(1) NOT NULL WITH DEFAULT	Indication of whether data pages are to be read in advance by prefetch: **'D'** Optimizer expects dynamic prefetch **'S'** Pure sequential prefetch **'L'** Prefetch through a page list **'U'** List prefetch with an unsorted RID list **blank** Unknown or no prefetch

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
COLUMN_FN_EVAL	CHAR(1) NOT NULL WITH DEFAULT	When an SQL aggregate function is evaluated: **'R'** While the data is being read from the table or index **'S'** While performing a sort to satisfy a GROUP BY clause **blank** After data retrieval and after any sorts
MIXOPSEQ	SMALLINT NOT NULL WITH DEFAULT	The sequence number of a step in a multiple index operation. **1, 2, ... *n*** For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, MU, DX, DI, or DU), the sequence number of the OR predicate in the SQL statement. (ACCESSTYPE is 'NR'). **0** For any other rows.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
ACCESS_DEGREE	SMALLINT	The number of parallel tasks or operations that are activated by a query. This value is determined at bind time; the actual number of parallel operations that are used at execution time could be different. This column contains 0 if a host variable is used. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
ACCESS_PGROUP_ID[2]	SMALLINT	The identifier of the parallel group for accessing the new table. A parallel group is a set of consecutive operations, executed in parallel, that have the same number of parallel tasks. This value is determined at bind time; it could change at execution time.This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
JOIN_DEGREE	SMALLINT	The number of parallel operations or tasks that are used in joining the composite table with the new table. This value is determined at bind time and can be 0 if a host variable is used. The actual number of parallel operations or tasks used at execution time could be different. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
JOIN_PGROUP_ID[2]	SMALLINT	The identifier of the parallel group for joining the composite table with the new table. This value is determined at bind time; it could change at execution time. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
SORTC_PGROUP_ID[3]	SMALLINT	The parallel group identifier for the parallel sort of the composite table. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
SORTN_PGROUP_ID[3]	SMALLINT	The parallel group identifier for the parallel sort of the new table. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
PARALLELISM_ MODE[2]	CHAR(1)	The kind of parallelism, if any, that is used at bind time: **C** Query CP parallelism. **I** Query I/O parallelism. Query I/O parallelism is deprecated and is likely to be removed in a future release. **X** Sysplex query parallelism. Sysplex query parallelism is deprecated and is likely to be removed in a future release. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns, if the method that it refers to does not apply, or if the plan or package was bound prior to DB2 10.
MERGE_ JOIN_ COLS	SMALLINT	The number of columns that are joined during a merge scan join (Method=2). This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
CORRELATION_ NAME	VARCHAR(128)	The correlation name of a table or view that is specified in the statement. If no correlation name exists, then the column is null. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
PAGE_RANGE	CHAR(1) NOT NULL WITH DEFAULT	Indication of whether the table qualifies for page range screening, so that plans scan only the partitions that are needed. **Y** Yes **blank** No
JOIN_TYPE	CHAR(1) NOT NULL WITH DEFAULT	The type of join: **F** FULL OUTER JOIN **L** LEFT OUTER JOIN **P** Pair-wise join **S** Star join **blank** INNER JOIN or no join RIGHT OUTER JOIN converts to a LEFT OUTER JOIN when you use it, so that JOIN_TYPE contains L.
GROUP_MEMBER	VARCHAR(24) NOT NULL WITH DEFAULT	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
IBM_ SERVICE_ DATA	VARCHAR(254) FOR BIT DATA	This column contains values that are for IBM use only.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
WHEN_OPTIMIZE	CHAR(1) NOT NULL WITH DEFAULT	When the access path was determined: **blank** At bind time, using a default filter factor for any host variables, parameter markers, or special registers. **B** At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is re-optimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE) must be specified for reoptimization to occur. **R** At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE) must be specified for this to occur.
| QBLOCK_TYPE[1]	CHAR(6) NOT NULL WITH DEFAULT	For each query block, an indication of the type of SQL operation that is performed. For the outermost query, this column identifies the statement type.5 on page 920
BIND_TIME	TIMESTAMP NOT NULL WITH DEFAULT	This column is deprecated. Use EXPLAIN_TIME instead.
OPTHINT	VARCHAR(128) NOT NULL WITH DEFAULT	A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
HINT_USED	VARCHAR(128) NOT NULL WITH DEFAULT	One of the following values: **'APREUSE'** When an access path was successfully reused because the APREUSE option was specified at bind or rebind. **'*opthint-value*'** When PLAN_TABLE access path hints are used. *opthint-value* is the value of the OPTHINT column for the hint that was used. **'SYSQUERYPLAN *query-id*'** When statement-level access path hints are used. *query-id* is the value of the QUERYID column in the SYSQUERYPLAN catalog table for the hint. **'EXPLAIN PACKAGE: COPY *copy-id*'** When the row is the result of an EXPLAIN PACKAGE statement. *copy-id* is one of the following values: **0** The current copy. **1** The previous copy. **2** The original copy.
PRIMARY_ ACCESSTYPE	CHAR(1) NOT NULL WITH DEFAULT	Indicates whether direct row access is attempted first: **'D'** DB2 tries to use direct row access with a rowid column. If DB2 cannot use direct row access with a rowid column at run time, it uses the access path that is described in the ACCESSTYPE column of PLAN_TABLE. **'T'** The base table or result file is materialized into a work file, and the work file is accessed via sparse index access. If a base table is involved, then ACCESSTYPE indicates how the base table is accessed. **blank** DB2 does not try to use direct row access by using a rowid column or sparse index access for a work file. The value of the ACCESSTYPE column of PLAN_TABLE provides information on the method of accessing the table.
PARENT_QBLOCKNO	SMALLINT NOT NULL WITH DEFAULT	A number that indicates the QBLOCKNO of the parent query block.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
TABLE_TYPE	CHAR(1)	The type of new table: **'B'** Buffers for SELECT from INSERT, SELECT from UPDATE, SELECT from MERGE, or SELECT from DELETE statement. **'C'** Common table expression **'F'** Table function **'I'** The new table is generated from an IN-LIST predicate. If the IN-LIST predicate is selected as the matching predicate, it will be accessed as an in-memory table. **'M'** Materialized query table **'Q'** Temporary intermediate result table (not materialized). For the name of a view or nested table expression, a value of Q indicates that the materialization was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed. **'R'** Recursive common table expression **'S'** Subquery (correlated or non-correlated) **'T'** Table **'W'** Work file The value of the column is null if the query uses GROUP BY, ORDER BY, or DISTINCT, which requires an implicit sort.
TABLE_ENCODE	CHAR(1) NOT NULL WITH DEFAULT	The encoding scheme of the table. The possible values are: **'A'** ASCII **'E'** EBCDIC **'U'** Unicode **'M'** The table contains multiple CCSID sets
TABLE_SCCSID	SMALLINT NOT NULL WITH DEFAULT	The SBCS CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
TABLE_MCCSID	SMALLINT NOT NULL WITH DEFAULT	The mixed CCSID value of the table. If the value of the TABLE_ENCODE column is M, the value is 0. If MIXED=NO in the application defaults module, the value is -2.
TABLE_DCCSID	SMALLINT NOT NULL WITH DEFAULT	The DBCS CCSID value of the table. If the value of the TABLE_ENCODE column is M, the value is 0. If MIXED=NO in the application defaults module, the value is -2.
ROUTINE_ID	INTEGER NOT NULL WITH DEFAULT	The values in this column are for IBM use only.

Table 184. Descriptions of columns in PLAN_TABLE (continued)

Column name	Data Type	Description
CTEREF	SMALLINT NOT NULL WITH DEFAULT	If the referenced table is a common table expression, the value is the top-level query block number.
STMTTOKEN	VARCHAR(240)	User-specified statement token.
PARENT_PLANNO	SMALLINT NOT NULL	Corresponds to the plan number in the parent query block where a correlated subquery is invoked. Or, for non-correlated subqueries, corresponds to the plan number in the parent query block that represents the work file for the subquery.
BIND_EXPLAIN_ONLY	CHAR(1) NOT NULL WITH DEFAULT	Identifies whether the row was inserted because a command specified the EXPLAIN(ONLY) option.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
EXPLAIN_TIME	TIMESTAMP NOT NULL WITH DEFAULT	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
MERGC	CHAR(1) NOT NULL WITH DEFAULT	Indicates whether the composite table is consolidated before the join. **'Y'** Yes **'N'** No
MERGN	CHAR(1) NOT NULL WITH DEFAULT	Indicates whether the new table is consolidated before the join **'Y'** Yes, the new table is consolidated before the join. **'N'** No, the new table is not consolidated before the join

Notes:

1. For PLAN_TABLE rows in which ACCESSTYPE='A' and QBLOCK_TYPE='SELECT', the values of all other columns except QUERYNO, APPLNAME, and PROGNAME are the default values for those columns.

2. In rows that are used for optimization hints, NULL values in the following columns indicate a hint for no parallelism:
   - PARALLELISM_MODE
   - ACCESS_PGROUP_ID
   - JOIN_PGROUP_ID
3. In rows that are used for optimization hints, NULL values in the following columns indicate a hint for no parallel sort:
   - SORTN_PGROUP_ID
   - SORTC_PGROUP_ID
4. The ACCESSTYPE column contains the following values:

   'A' The query is sent to an accelerator server.

   'DI' By an intersection of multiple DOCID lists to return the final DOCID list

   'DU' By a union of multiple DOCID lists to return the final DOCID list

   'DX' By an XML index scan on the index that is named in ACCESSNAME to return a DOCID list

   'E' By direct row access using a row change timestamp column.

   'H' By hash access. IF an overflow condition occurs, the hash overflow index that is identified by ACCESSCREATOR and ACCESSNAME is used.

   'HN' By hash access using an IN predicate, or an IN predicate that DB2 generates. If a hash overflow condition occurs, the hash overflow index that is identified in ACCESSCREATOR and ACCESSNAME is used.

   'I' By an index (identified in ACCESSCREATOR and ACCESSNAME)

   'IN' By an index scan when the matching predicate contains an IN predicate and the IN-list is accessed through an in-memory table.

   'I1' By a one-fetch index scan

   'M' By a multiple index scan. A row that contains this value might be followed by a row that contains one of the following values:
   - 'DI'
   - 'DU'
   - 'MH'
   - 'MI'
   - 'MU'
   - 'MX'

   'MH' By the hash overflow index named in ACCESSNAME. A row that contains this value always follows a row that contains M.

   'MI' By an intersection of multiple indexes. A row that contains this value always follows a row that contains M.

   'MU' By a union of multiple indexes. A row that contains this value always follows a row that contains M.

   'MX' By an index scan on the index named in ACCESSNAME. When the access method MX follows the access method DX, DI, or DU, the table is accessed by the DOCID index by using the DOCID list that is returned by DX, DI, or DU. A row that contains this value always follows a row that contains M.

   'N' One of the following types:
   - By an index scan when the matching predicate contains the IN keyword
   - By an index scan when DB2 rewrites a query using the IN keyword

   'O' By a work file scan, as a result of a correlated subquery.

   'NR' Range list access.

   'P' By a dynamic pair-wise index scan

   'R' By a table space scan

'RW' By a work file scan of the result of a materialized user-defined table function

'V' By buffers for an INSERT statement within a SELECT

blank Not applicable to the current row

5. The QBLOCK_TYPE column contains the following values:

'SELECT'
SELECT

'INSERT'
INSERT

'UPDATE'
UPDATE

'MERGE'
MERGE

'DELETE'
DELETE

'SELUPD'
SELECT with FOR UPDATE OF

DELCUR
DELETE WHERE CURRENT OF CURSOR

'UPDCUR'
UPDATE WHERE CURRENT OF CURSOR

'CORSUB'
Correlated subselect or fullselect

'TRUNCA'
TRUNCATE

'NCOSUB'
Noncorrelated subselect or fullselect

'TABLEX'
Table expression

'TRIGGR'
WHEN clause on CREATE TRIGGER

'UNION'
UNION

'UNIONA'
UNION ALL

'INTERS'
INTERSECT

'INTERA'
INTERSECT ALL

'EXCEPT'
EXCEPT

'EXCEPTA'
EXCEPT ALL

'PRUNED'
DB2 does not generate an access path for the query because the query

is guaranteed to qualify zero rows, such as the case of an always-false WHERE clause. For example:`WHERE 0=1`

6. DB2 inserts a value into the REMARKS column at bind or rebind when the EXPLAIN(ONLY) option is specified and reuse or comparison fails for an access path. The value might include the following information:
   - A reason code that corresponds to the reason codes in SQLCODE +395 when reuse fails
   - The name of the unmatched PLAN_TABLE column for which comparison failed
   - A string that identifies that unmatched rows where found

The PLAN_TABLE_HINT_IX index

The PLAN_TABLE_HINT_IX index improves prepare performance when access path hints are used. This index is required for all types of statement-level optimization hints. The PLAN_TABLE_HINT_IX index is optional, although strongly recommended, for PLAN_TABLE access path hints.

The statement that creates the PLAN_TABLE_HINT_IX index is included as part of the DSNTESC member of the SDSNSAMP library. PSPI

Related concepts:

Interpreting data access by using EXPLAIN

Related tasks:

Preparing to influence access paths

Generating visual representations of access plans (IBM Data Studio)

Related reference:

Support Portal: IBM DB2 Analytics Accelerator for z/OS

Format of PLAN_TABLE in DB2 10 (DB2 for z/OS What's New?)

DSN_COLDIST_TABLE

The column distribution table contains non-uniform column group statistics that are obtained dynamically by DB2 from non-index leaf pages.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2

and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'user-ID'

You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

PSPI

The following table shows the descriptions of the columns in the DSN_COLDIST_TABLE table.

Table 185. Descriptions of columns in DSN_COLDIST_TABLE

Column name	Data Type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.

Table 185. Descriptions of columns in DSN_COLDIST_TABLE (continued)

Column name	Data Type	Description
APPLNAME	VARCHAR(128) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
GROUP_MEMBER	VARCHAR(128) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.

Table 185. Descriptions of columns in DSN_COLDIST_TABLE (continued)

Column name	Data Type	Description
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
SCHEMA	VARCHAR(128) NOT NULL	The schema of the table that contains the column.
TBNAME	VARCHAR(128) NOT NULL	The name of the table that contains the column.
NAME	VARCHAR(128) NOT NULL	Name of the column. If the value of NUMCOLUMNS is greater than 1, this name identifies the first column name of the set of columns associated with the statistics.

Table 185. Descriptions of columns in DSN_COLDIST_TABLE (continued)

Column name	Data Type	Description
COLVALUE	VARCHAR(2000) NOT NULL FOR BIT DATA	Contains the data of a frequently occurring value in the column. Statistics are not collected for an index on a ROWID column. If the value has a non-character data type, the data might not be printable. This column might contain values that depend on the value of the type column: **TYPE='T'** One of the following values: - 'E3C2C1C3C1D9C4C6' for TBACARDF - 'E3C2C1D5C1C3E3C6' for TBANPAGF - 'E3C2C1D5D7C1C7C6' for TBANACTF **TYPE='L'** 'C3C1E3C6D3C4C3C6' for CATFLDCF **TYPE='P'** One of the following values: - 'D7C3C1D7D5D9E6C6' for PCAPNRWF - 'D7C3C1D7D5D7C7C6' for PCAPNPGF
TYPE	CHAR(1) NOT NULL	The type of statistics: **C** Cardinality **F** Frequent value **H** Histogram **T** Real-time table cardinality **L** Real-time column cardinality (unique index only) **P** real-time partition cardinality
CARDF	FLOAT NOT NULL	For TYPE='C', the number of distinct values for the column group. For TYPE='H', the number of distinct values for the column group in a quantile indicated by the value of the QUANTILENO column. For TYPE='T', a value related to real-time statistics table values that are determined by the COLVALUE column. For TYPE= 'L', a value related to a real-time statistics column value that is determined by the COLVALUE column. The QUANTILENO column contains the column number. The NAME column contains the column name. For TYPE='P' a value related to real-time statistics partition value that is determined by the COLVALUE column. The QUANTILENO column contains the partition number.

Table 185. Descriptions of columns in DSN_COLDIST_TABLE (continued)

Column name	Data Type	Description
COLGROUPCOLNO	VARCHAR(254) NOT NULL FOR BIT DATA	The identity of the set of columns associated with the statistics. If the statistics are only associated with a single column, the field contains a zero length. Otherwise, the field is an array of SMALLINT column numbers with a dimension equal to the value in the NUMCOLUMNS column. This is an updatable column.
NUMCOLUMNS	SMALLINT NOT NULL	Identifies the number of columns associated with the statistics.
FREQUENCYF	FLOAT NOT NULL	The percentage of rows in the table with the value that is specified in the COLVALUE column when the number is multiplied by 100. For example, a value of '1' indicates 100%. A value of '.153' indicates 15.3%.
QUANTILENO	SMALLINT NOT NULL	The ordinary sequence number of a quantile in the whole consecutive value range, from low to high. This column is not updatable. For TYPE= 'L', this column contains the column number. For TYPE='P', the column contains the partition number.
LOWVALUE	VARCHAR(2000) NOT NULL FOR BIT DATA	For TYPE='H', this is the lower bound for the quantile indicated by the value of the QUANTILENO column. Not used if the value of the TYPE column is not 'H'. This column is not updatable.
HIGHVALUE	VARCHAR(2000) NOT NULL FOR BIT DATA	For TYPE='H', this is the higher bound for the quantile indicated by the value of the QUANTILENO column. This column is not used if the value of the TYPE column is not 'H'. This column is not updatable.

PSPI

Related concepts:

Dynamic collection of index filtering estimates

DSN_DETCOST_TABLE

The detailed cost table, DSN_DETCOST_TABLE, contains information about detailed cost estimation of the mini-plans in a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_DETCOST_TABLE.

Table 186. DSN_DETCOST_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each mini-plan with a query block.
OPENIO	FLOAT(4) NOT NULL	The Do-at-open IO cost for non-correlated subquery.
OPENCPU	FLOAT(4) NOT NULL	The Do-at-open CPU cost for non-correlated subquery.
OPENCOST	FLOAT(4) NOT NULL	The Do-at-open total cost for non-correlated subquery.
DMIO	FLOAT(4) NOT NULL	IBM internal use only.
DMCPU	FLOAT(4) NOT NULL	IBM internal use only.
DMTOT	FLOAT(4) NOT NULL	IBM internal use only.
SUBQIO	FLOAT(4) NOT NULL	IBM internal use only.
SUBQCOST	FLOAT(4) NOT NULL	IBM internal use only.
BASEIO	FLOAT(4) NOT NULL	IBM internal use only.
BASECPU	FLOAT(4) NOT NULL	IBM internal use only.
BASETOT	FLOAT(4) NOT NULL	IBM internal use only.
ONECOMPROWS	FLOAT(4) NOT NULL	The number of rows qualified after applying local predicates.
IMLEAF	FLOAT(4) NOT NULL	The number of index leaf pages scanned by Data Manager.
IMIO	FLOAT(4) NOT NULL	IBM internal use only.
IMPREFH	CHAR(2) NOT NULL	IBM internal use only.
IMMPRED	INTEGER NOT NULL	IBM internal use only.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
IMFF	FLOAT(4) NOT NULL	The filter factor of matching predicates only.
IMSRPRED	INTEGER NOT NULL	IBM internal use only.
IMFFADJ	FLOAT(4) NOT NULL	The filter factor of matching and screening predicates.
IMSCANCST	FLOAT(4) NOT NULL	IBM internal use only.
IMROWCST	FLOAT(4) NOT NULL	IBM internal use only.
IMPAGECST	FLOAT(4) NOT NULL	IBM internal use only.
IMRIDSORT	FLOAT(4) NOT NULL	IBM internal use only.
IMMERGCST	FLOAT(4) NOT NULL	IBM internal use only.
IMCPU	FLOAT(4) NOT NULL	IBM internal use only.
IMTOT	FLOAT(4) NOT NULL	IBM internal use only.
IMSEQNO	SMALLINT NOT NULL	IBM internal use only.
DMPEREFH	FLOAT(4) NOT NULL	IBM internal use only.
DMCLUDIO	FLOAT(4) NOT NULL	IBM internal use only.
DMPREDS	INTEGER NOT NULL	IBM internal use only.
DMSROWS	FLOAT(4) NOT NULL	IBM internal use only.
DMSCANCST	FLOAT(4) NOT NULL	IBM internal use only.
DMCOLS	FLOAT(4) NOT NULL	The number of data manager columns.
DMROWS	FLOAT(4) NOT NULL	The number of data manager rows returned (after all stage 1 predicates are applied).
RDSROWCST	FLOAT(4) NOT NULL	IBM internal use only.
DMPAGECST	FLOAT(4) NOT NULL	IBM internal use only.
DMDATAIO	FLOAT(4) NOT NULL	IBM internal use only.
DMDATAIO	FLOAT(4) NOT NULL	IBM internal use only.
DMDATACPU	FLOAT(4) NOT NULL	IBM internal use only.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
DMDATACPU	FLOAT(4) NOT NULL	IBM internal use only.
RDSROW	FLOAT(4) NOT NULL	The number of RDS rows returned (after all stage 1 and stage 2 predicates are applied).
SNCOLS	SMALLINT NOT NULL	The number of columns as sort input for new table.
SNROWS	FLOAT(4) NOT NULL	The number of rows as sort input for new table.
SNRECSZ	INTEGER NOT NULL	The record size for new table.
SNPAGES	FLOAT(4) NOT NULL	The page size for new table.
SNRUNS	FLOAT(4) NOT NULL	The number of runs generated for sort of new table.
SNMERGES	FLOAT(4) NOT NULL	The number of merges needed during sort.
SNIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCSCANIO	FLOAT(4) NOT NULL	IBM internal use only.
SNSCANCPU	FLOAT(4) NOT NULL	IBM internal use only.
SNCCOLS	FLOAT(4) NOT NULL	The number of columns as sort input for Composite table.
SCROWS	FLOAT(4) NOT NULL	The number of rows as sort input for Composite Table.
SCRECSZ	FLOAT(4) NOT NULL	The record size for Composite table.
SCPAGES	FLOAT(4) NOT NULL	The page size for Composite table.
SCRUNS	FLOAT(4) NOT NULL	The number of runs generated during sort of composite.
SCMERGES	FLOAT(4) NOT NULL	The number of merges needed during sort of composite.
SCIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANIO	FLOAT(4) NOT NULL	IBM internal use only.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
SCSCANCPU	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCARD	FLOAT(4) NOT NULL	The total composite cardinality.
COMPIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCOST	FLOAT(4) NOT NULL	The total cost.
JOINCOLS	SMALLINT NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
COSTBLK	INTEGER NOT NULL	IBM internal use only.
COSTSTOR	INTEGER NOT NULL	IBM internal use only.
MPBLK	INTEGER NOT NULL	IBM internal use only.
MPSTOR	INTEGER NOT NULL	IBM internal use only.
COMPOSITES	INTEGER NOT NULL	IBM internal use only.
CLIPPED	INTEGER NOT NULL	IBM internal use only.
TABREF	VARCHAR(64) NOT NULL FOR BIT DATA	IBM internal use only.
MAX_COMPOSITES	INTEGER NOT NULL	IBM internal use only.
MAX_STOR	INTEGER NOT NULL	IBM internal use only.
MAX_CPU	INTEGER NOT NULL	IBM internal use only.
MAX_ELAP	INTEGER NOT NULL	IBM internal use only.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
TBL_JOINED_THRESH	INTEGER NOT NULL	IBM internal use only.
STOR_USED	INTEGER NOT NULL	IBM internal use only.
CPU_USED	INTEGER NOT NULL	IBM internal use only.
ELAPSED	INTEGER NOT NULL	IBM internal use only.
MIN_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
PSEQIOCOST	FLOAT(4) NOT NULL	IBM internal use only.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
PSEQIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJCOST	FLOAT(4) NOT NULL	IBM internal use only.
UNCERTAINTY	FLOAT(4) NOT NULL WITH DEFAULT	Describes the uncertainty factor of inner table index access. It is aggregated from uncertainty of inner table probing predicates. A larger value indicates a higher uncertainty. 0 indicates no uncertainty or uncertainty not considered.
UNCERTAINTY_1T	FLOAT(4) NOT NULL WITH DEFAULT	Describes the uncertainty factor of ONECOMPROWS column of the table. It is aggregated from all local predicates on the table. A larger value indicates a higher uncertainty. 0 indicates no uncertainty or uncertainty not considered.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(128) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.

Table 186. DSN_DETCOST_TABLE description (continued)

Column name	Data type	Description
IMNP	FLOAT(4) NOT NULL WITH DEFAULT	IBM internal use only.
DMNP	FLOAT(4) NOT NULL WITH DEFAULT	IBM internal use only.
IMJC	FLOAT(4) NOT NULL WITH DEFAULT	IBM internal use only.
IMFC	FLOAT(4) NOT NULL WITH DEFAULT	IBM internal use only.
IMJBC	FLOAT(4) NOT NULL WITH DEFAULT	IBM internal use only.
IMJFC	FLOAT(4) NOT NULL WITH DEFAULT	IBM internal use only.
CRED	INTEGER NOT NULL WITH DEFAULT	IBM internal use only.

PSPI

Related reference:

Support Portal: IBM DB2 Analytics Accelerator for z/OS

DSN_FILTER_TABLE

The filter table, DSN_FILTER_TABLE, contains information about how predicates are used during query processing.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'

You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_FILTER_TABLE.

Table 187. DSN_FILTER_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
PLANNO	SMALLINT	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.

Table 187. DSN_FILTER_TABLE description (continued)

Column name	Data type	Description
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
ORDERNO	INTEGER NOT NULL	The sequence number of evaluation. Indicates the order in which the predicate is applied within each stage
PREDNO	INTEGER NOT NULL	The predicate number, a number used to identify a predicate within a query.
STAGE	CHAR(9) NOT NULL	The processing stage in which the predicate is evaluated: **MATCHING** During the index matching stage. **SCREENING** During the index screening stage. **STAGE1** During stage 1 processing, after data page access. **STAGE2** During stage 2 processing on the returned data rows.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.

Table 187. DSN_FILTER_TABLE description (continued)

Column name	Data type	Description
MIXOPSEQNO	SMALLINT NOT NULL	IBM internal use only.
REEVAL	CHAR(1) NOT NULL	IBM internal use only.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
PUSHDOWN	CHAR(1) NOT NULL WITH DEFAULT	Whether the predicate is pushed down the Index Manager or Data Manager subcomponents for evaluation: **'I'** The Index Manager subcomponent evaluates the predicate. **'D'** The Data Manager subcomponent evaluates the predicate. **blank** The predicate is not pushed down for evaluation.

PSPI

DSN_FUNCTION_TABLE

The function table, DSN_FUNCTION_TABLE, contains descriptions of functions that are used in specified SQL statements.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2

and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'user-ID'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

PSPI

The following table describes the columns of DSN_FUNCTION_TABLE.

Table 188. Descriptions of columns in DSN_FUNCTION_TABLE

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL WITH DEFAULT	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	INTEGER NOT NULL WITH DEFAULT	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.

Table 188. Descriptions of columns in DSN_FUNCTION_TABLE (continued)

Column name	Data type	Description
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
GROUP_MEMBER	VARCHAR(24) NOT NULL WITH DEFAULT	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
EXPLAIN_TIME	TIMESTAMP NOT NULL WITH DEFAULT	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
SCHEMA_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	The schema name of the function invoked in the explained statement.
FUNCTION_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the function invoked in the explained statement.
SPEC_FUNC_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	The specific name of the function invoked in the explained statement.
FUNCTION_TYPE	CHAR(2) NOT NULL WITH DEFAULT	The type of function invoked in the explained statement. Possible values are: **CU** Column function **SU** Scalar function **TU** Table function

Table 188. Descriptions of columns in DSN_FUNCTION_TABLE (continued)

Column name	Data type	Description
VIEW_CREATOR	VARCHAR(128) NOT NULL WITH DEFAULT	If the function specified in the FUNCTION_NAME column is referenced in a view definition, the creator of the view. Otherwise, blank.
VIEW_NAME	VARCHAR(128) NOT NULL WITH DEFAULT	If the function specified in the FUNCTION_NAME column is referenced in a view definition, the name of the view. Otherwise, blank.
PATH	VARCHAR(2048) NOT NULL WITH DEFAULT	The value of the SQL path that was used to resolve the schema name of the function.
FUNCTION_TEXT	VARCHAR(1500) NOT NULL WITH DEFAULT	The text of the function reference (the function name and parameters). If the function reference is over 100 bytes, this column contains the first 100 bytes. For functions specified in infix notation, FUNCTION_TEXT contains only the function name. For example, for a function named /, which overloads the SQL divide operator, if the function reference is A/B, FUNCTION_TEXT contains only /.
FUNC_VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	For a version of a non-inline SQL scalar function, this column contains the version identifier. For all other cases, this column contains a zero length string. A version of a non-inline SQL scalar function is defined in the SYSIBM.SYSROUTINES table with ORIGIN='Q', FUNCTION_TYPE='S', INLINE='N', and VERSION column containing the version identifier.
SECURE	CHAR(1) NOT NULL WITH DEFAULT	Whether the user-defined function is secure.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

Related tasks:

Checking how DB2 resolves functions by using DSN_FUNCTION_TABLE (DB2 Application programming and SQL)

DSN_KEYTGTDIST_TABLE

The key-target distribution table contains non-uniform index expression statistic that are obtained dynamically by the DB2 optimizer.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

COLUMN descriptions

The following table shows the descriptions of the columns in the DSN_KEYTGTDIST_TABLE table.

Table 189. Descriptions of columns in DSN_KEYTGTDIST_TABLE

Column name	Data Type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
APPLNAME	VARCHAR(128) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.

Table 189. Descriptions of columns in DSN_KEYTGTDIST_TABLE (continued)

Column name	Data Type	Description
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
GROUP_MEMBER	VARCHAR(128) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

Table 189. Descriptions of columns in DSN_KEYTGTDIST_TABLE (continued)

Column name	Data Type	Description
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
IXSCHEMA	VARCHAR(128) NOT NULL	The qualifier of the index.
IXNAME	VARCHAR(128) NOT NULL	The name of the index.
KEYSEQ	VARCHAR(128) NOT NULL	The numeric position of the key-target in the index.
KEYVALUE	VARCHAR(2000) NOT NULL FOR BIT DATA	Contains the data of a frequently occurring value. Statistics are not collected for an index on a ROWID column. If the value has a non-character data type, the data might not be printable. When the value of the TYPE column contains 'I', this column contains one of the following values: • 'C9C4E7C6E4D3D2C6' for IDXFULKF • 'C9C4E7D3C5C1C6C6' for IDXLEAFF • 'C9C4E7D5D3E5D3C6' for IDXNLVLF
TYPE	CHAR(1) NOT NULL	The type of statistics: **C** Cardinality **F** Frequent value **H** Histogram **I** Real-time index statistics
CARDF	FLOAT NOT NULL	For TYPE='C', the number of distinct values for the column group. For TYPE='H', the number of distinct values for the column group in a quantile indicated by the value of the QUANTILENO column. For TYPE='I', a value related to real-time index statistics values determined by the KEYVALUE column.
KEYGROUPKEYNO	VARCHAR(254) NOT NULL FOR BIT DATA	Contains a value that identifies the set of keys that are associated with the statistics. If the statistics are associated with more than a single key, it contains an array of SMALLINT key numbers with a dimension that is equal to the value in NUMKEYS. If the statistics are only associated with a single key, it contains 0.
NUMKEYS	SMALLINT NOT NULL	The number of keys that are associated with the statistics.

Table 189. Descriptions of columns in DSN_KEYTGTDIST_TABLE (continued)

Column name	Data Type	Description
FREQUENCYF	FLOAT NOT NULL	The percentage of rows in the table with the value that is specified in the COLVALUE column when the number is multiplied by 100. For example, a value of '1' indicates 100%. A value of '.153' indicates 15.3%.
QUANTILENO	SMALLINT NOT NULL	The ordinary sequence number of a quantile in the whole consecutive value range, from low to high. This column is not updatable
LOWVALUE	VARCHAR(2000) NOT NULL FOR BIT DATA	For TYPE='H', this is the lower bound for the quantile indicated by the value of the QUANTILENO column. Not used if the value of the TYPE column is not 'H'. This column is not updatable.
HIGHVALUE	VARCHAR(2000) NOT NULL FOR BIT DATA	For TYPE='H', this is the higher bound for the quantile indicated by the value of the QUANTILENO column. This column is not used if the value of the TYPE column is not 'H'. This column is not updatable.

PSPI

Related concepts:

Dynamic collection of index filtering estimates

DSN_PGRANGE_TABLE

The page range table, DSN_PGRANGE_TABLE, contains information about qualified partitions for all page range scans in a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'

One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'

You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_PGRANGE_TABLE.

Table 190. DSN_PGRANGE_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
TABNO	SMALLINT NOT NULL	The table number, a number which uniquely identifies the corresponding table reference within a query.
RANGE	SMALLINT NOT NULL	The sequence number of the current page range.
FIRSTPART	SMALLINT NOT NULL	The starting partition in the current page range.
LASTPART	SMALLINT NOT NULL	The ending partition in the current page range.
NUMPARTS	SMALLINT NOT NULL	The number of partitions in the current page range.

Table 190. DSN_PGRANGE_TABLE description (continued)

Column name	Data type	Description
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.

Table 190. DSN_PGRANGE_TABLE description (continued)

Column name	Data type	Description
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

DSN_PGROUP_TABLE

The parallel group table, DSN_PGROUP_TABLE, contains information about the parallel groups in a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_PGROUP_TABLE

Table 191. DSN_PGROUP_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
PLANNAME	VARCHAR(24) NOT NULL	The application plan name.
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.

Table 191. DSN_PGROUP_TABLE description (continued)

Column name	Data type	Description
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
GROUPID	SMALLINT NOT NULL	The parallel group identifier within the current query block.
FIRSTPLAN	SMALLINT NOT NULL	The plan number of the first contributing mini-plan associated within this parallel group.
LASTPLAN	SMALLINT NOT NULL	The plan number of the last mini-plan associated within this parallel group.
CPUCOST	REAL NOT NULL	The estimated total CPU cost of this parallel group in milliseconds.
IOCOST	REAL NOT NULL	The estimated total I/O cost of this parallel group in milliseconds.
BESTTIME	REAL NOT NULL	The estimated elapsed time for each parallel task for this parallel group.
DEGREE	SMALLINT NOT NULL	The degree of parallelism for this parallel group determined at bind time. Max parallelism degree if the Table space is large is 255, otherwise 64.

Table 191. DSN_PGROUP_TABLE description (continued)

Column name	Data type	Description
MODE	CHAR(1) NOT NULL	The parallel mode: **'I'** I/O parallelism **'C'** CPU parallelism **'X'** Multiple CPU Sysplex parallelism (highest level) **'N'** No parallelism
REASON	SMALLINT NOT NULL	The reason code for downgrading parallelism mode.
LOCALCPU	SMALLINT NOT NULL	The number of CPUs currently online when preparing the query.
TOTALCPU	SMALLINT NOT NULL	The total number of CPUs in Sysplex. LOCALCPU and TOTALCPU are different only for the DB2 coordinator in a Sysplex.
FIRSTBASE	SMALLINT	The table number of the table that partitioning is performed on.
LARGETS	CHAR(1)	'Y' if the TableSpace is large in this group.
PARTKIND	CHAR(1)	The partitioning type: **'L'** Logical partitioning **'P'** Physical partitioning
GROUPTYPE	CHAR(3)	Determines what operations this parallel group contains: table Access, Join, or Sort 'A' 'AJ' 'AJS'
ORDER	CHAR(1)	The ordering requirement of this parallel group : **'N'** No order. Results need no ordering. **'T'** Natural Order. Ordering is required but results already ordered if accessed via index. **'K'** Key Order. Ordering achieved by sort. Results ordered by sort key. This value applies only to parallel sort.
STYLE	CHAR(4)	The Input/Output format style of this parallel group. Blank for IO Parallelism. For other modes: **'RIRO'** Records IN, Records OUT **'WIRO'** Work file IN, Records OUT **'WIWO'** Work file IN, Work file OUT

Table 191. DSN_PGROUP_TABLE description (continued)

Column name	Data type	Description
RANGEKIND	CHAR(1)	The range type: **'K'** Key range **'L'** IN-list elements partitioning **'P'** Page range **'R'** Record range partitioning
NKEYCOLS	SMALLINT	The number of interesting key columns, that is, the number of columns that will participate in the key operation for this parallel group.
LOWBOUND	VARCHAR(40) FOR BIT DATA	The low bound of parallel group.
HIGHBOUND	VARCHAR(40) FOR BIT DATA	The high bound of parallel group.
LOWKEY	VARCHAR(40) FOR BIT DATA	The low key of range if partitioned by key range.
HIGHKEY	VARCHAR(40) FOR BIT DATA	The high key of range if partitioned by key range.
FIRSTPAGE	CHAR(4) FOR BIT DATA	The first page in range if partitioned by page range.
LASTPAGE	CHAR(4) FOR BIT DATA	The last page in range if partitioned by page range.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
HOST_REASON	SMALLINT	IBM internal use only.
PARA_TYPE	CHAR(4)	IBM internal use only.
PART_INNER	CHAR(1)	IBM internal use only.
GRNU_KEYRNG	CHAR(1)	IBM internal use only.
OPEN_KEYRNG	CHAR(1)	IBM internal use only.
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
STRAW_MODEL	CHAR(1) NOT NULL WITH DEFAULT	IBM internal use only.

PSPI

DSN_PREDICAT_TABLE

The predicate table, DSN_PREDICAT_TABLE, contains information about all of the predicates in a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'

One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'

You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of the DSN_PREDICAT_TABLE

Table 192. DSN_PREDICAT_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
PREDNO	INTEGER NOT NULL	The predicate number, a number used to identify a predicate within a query.

Table 192. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
TYPE	CHAR(8) NOT NULL	A string used to indicate the type or the operation of the predicate. The possible values are: • 'AND' • 'BETWEEN' • 'EQUAL' • 'EXISTS • 'HAVING' • 'IN' • 'LIKE' • 'NOT LIKE' • 'NOTEXIST' • 'OTHERS' • 'OR' • 'RANGE' • 'SUBQUERY' • 'XEXISTS' • 'NXEXISTS'
LEFT_HAND_SIDE	VARCHAR(128) NOT NULL	Describes the left side of the predicate. If the left side of the predicate is a table column, this value indicates the name of that column. Other possible values are: • 'VALUE' • 'COLEXP' • 'NONCOLEXP' • 'CORSUB' • 'NONCORSUB' • 'SUBQUERY' • 'EXPRESSION' • Blanks
LEFT_HAND_PNO	INTEGER NOT NULL	If the predicate is a compound predicate (AND/OR), then this column indicates the first child predicate. However, this column is not reliable when the predicate tree consolidation happens. Use PARENT_PNO instead to reconstruct the predicate tree.
LHS_TABNO	SMALLINT NOT NULL	If the left side of the predicate is a table column or a column expression in an expression-based index, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
LHS_QBNO	SMALLINT NOT NULL	If the left side of the predicate is a table column or a column expression in expression-based index, then this column indicates a number which uniquely identifies the corresponding query block within a query.

Table 192. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
RIGHT_HAND_SIDE	VARCHAR(128) NOT NULL	Describes the right side of the predicate. If the right side of the predicate is a table column, this value column indicates the column name. Other possible values are: • 'VALUE' • 'COLEXP' • 'NONCOLEXP' • 'CORSUB' • 'NONCORSUB' • 'SUBQUERY' • 'EXPRESSION' • Blanks
RIGHT_HAND_PNO	INTEGER NOT NULL	If the predicate is a compound predicate (AND/OR), then this column indicates the second child predicate. However, this column is not reliable when the predicate tree consolidation happens. Use PARENT_PNO instead to reconstruct the predicate tree.
RHS_TABNO	CHAR(1) NOT NULL	If the right side of the predicate is a table column or a column expression in an index on expression, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
RHS_QBNO	CHAR(1) NOT NULL	If the right side of the predicate is a subquery or a column expression in an expression-based index, then this column indicates a number which uniquely identifies the corresponding query block within a query.
FILTER_FACTOR	FLOAT NOT NULL	The estimated filter factor.
BOOLEAN_TERM	CHAR(1) NOT NULL	Whether this predicate can be used to determine the truth value of the whole WHERE clause.
SEARCHARG	CHAR(1) NOT NULL	Whether this predicate can be processed by data manager (DM). If it is not, then the relational data service (RDS) needs to be used to take care of it, which is more costly.
JOIN	CHAR(1) NOT NULL	Whether the predicate can be used as a simple join predicate between two tables.
AFTER_JOIN	CHAR(1) NOT NULL	Indicates the predicate evaluation phase: **'A'** After join **'D'** During join **blank** Not applicable
ADDED_PRED	CHAR(1) NOT NULL	Whether it is generated by transitive closure, which means DB2 can generate additional predicates to provide more information for access path selection, when the set of predicates that belong to a query logically imply other predicates.
REDUNDANT_PRED	CHAR(1) NOT NULL	Whether it is a redundant predicate, which means evaluation of other predicates in the query already determines the result that the predicate provides.
DIRECT_ACCESS	CHAR(1) NOT NULL	Whether the predicate is direct access, which means one can navigate directly to the row through ROWID.
KEYFIELD	CHAR(1) NOT NULL	Whether the predicate includes the index key column of the involved table for all applicable indexes considered by DB2.

Table 192. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
CATEGORY	SMALLINT NOT NULL	IBM internal use only.
CATEGORY_B	SMALLINT NOT NULL	IBM internal use only.
TEXT	VARCHAR(2000) NOT NULL	The text of the transformed predicate text. If the text of the predicate contains more than 2000 characters, it is truncated.
PRED_ENCODE	CHAR(1) NOT NULL WITH DEFAULT	IBM internal use only.
PRED_CCSID	SMALLINT NOT NULL WITH DEFAULT	IBM internal use only.
PRED_MCCSID	SMALLINT NOT NULL WITH DEFAULT	IBM internal use only.
MARKER	CHAR(1) NOT NULL WITH DEFAULT	Whether this predicate includes host variables, parameter markers, or special registers.
PARENT_PNO	INTEGER NOT NULL	The parent predicate number. If this predicate is a root predicate within a query block, then this column is 0.
NEGATION	CHAR(1) NOT NULL	Whether this predicate is negated via NOT.
LITERALS	VARCHAR(128) NOT NULL	This column indicates the literal value or literal values separated by colon symbols.
CLAUSE	CHAR(8) NOT NULL	The clause where the predicate exists: **'HAVING '** The HAVING clause **'ON '** The ON clause **'WHERE '** The WHERE clause **SELECT** The SELECT clause
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

Table 192. DSN_PREDICAT_TABLE description (continued)

Column name	Data type	Description
ORIGIN	CHAR(1) NOT NULL WITH DEFAULT	Indicates the origin of the predicate. **Blank** Generated by DB2 **C** Column mask **R** Row permission **U** Specified by the user
UNCERTAINTY	FLOAT(4) NOT NULL WITH DEFAULT	Describes the uncertainty factor of a predicate's estimated filter factor. A bigger value indicates a higher degree of uncertainty. Value zero indicates no uncertainty or uncertainty not considered.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

Related concepts:

Predicates (DB2 SQL)

Predicates and access path selection

DSN_PTASK_TABLE

The parallel tasks table, DSN_PTASK_TABLE, contains information about all of the parallel tasks in a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
: One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
: You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_PTASK_TABLE.

Table 193. DSN_PTASK_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
PGDNO	SMALLINT NOT NULL	The parallel group identifier within the current query block. This value corresponds to the value of the GROUPID column in DSN_PGROUP_TABLE table rows.
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
LPTNO	SMALLINT NOT NULL	The parallel task number.
KEYCOLID	SMALLINT	The key column ID (KEY range only).
DPSI	CHAR(1) NOT NULL	Indicates if a data partition secondary index (DPSI) is used.
LPTLOKEY	VARCHAR(40) FOR BIT DATA	The low key value for this key column for this parallel task (KEY range only).

Table 193. DSN_PTASK_TABLE description (continued)

Column name	Data type	Description
LPTHIKEY	VARCHAR(40) FOR BIT DATA	The high key value for this key column for this parallel task (KEY range only).
LPTLOPAG	CHAR(4) FOR BIT DATA	The low page information if partitioned by page range.
LPTLHIPAG	CHAR(4) FOR BIT DATA	The high page information if partitioned by page range.
LPTLOPG[1]	CHAR(4) FOR BIT DATA	The lower bound page number for this parallel task (Page range or DPSI enabled only).
LPTHIPG[1]	CHAR(4) FOR BIT DATA	The upper bound page number for this parallel task (Page range or DPSI enabled only).
LPTLOPT[1]	SMALLINT	The lower bound partition number for this parallel task (Page range or DPSI enabled only).
LPTHIPT[1]	SMALLINT	The upper bound partition number for this parallel task (Page range or DPSI enabled only).
KEYCOLDT	SMALLINT	The data type for this key column (KEY range only).
KEYCOLPREC	SMALLINT	The precision/length for this key column (KEY range only).
KEYCOLSCAL	SMALLINT	The scale for this key column (KEY range with Decimal datatype only).
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.

Table 193. DSN_PTASK_TABLE description (continued)

Column name	Data type	Description
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

Notes:

1. The name of these columns originally contained the # symbol as the last character in the names. However, the names that contain these characters are obsolete and are no longer supported.

PSPI

DSN_QUERYINFO_TABLE

The query information table, DSN_QUERYINFO_TABLE, contains information about the eligibility of query blocks for automatic query rewrite, information about the materialized query tables that are considered for eligible query blocks, reasons why ineligible query blocks are not eligible, and information about acceleration of query blocks.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

userID You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind, or rebind, a plan or package with the EXPLAIN(YES) option. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

Table 194. Descriptions of columns in DSN_QUERYINFO_TABLE

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
QINAME1	VARCHAR(128) NOT NULL WITH DEFAULT	When TYPE='A': • When REASON_CODE=0, this value is the name of the accelerator server to which the statement is sent. • When REASON_CODE<>0, the statement was not sent to an accelerator server. The REASON_CODE value indicates why the statement was not sent to the accelerator server. When TYPE='ACCELMDL', this statement used accelerator modeling.
QINAME2	VARCHAR(128) NOT NULL WITH DEFAULT	When TYPE='A' and REASON_CODE=0, this value is the name of the location name of the accelerator server to which the statement is sent.

Table 194. Descriptions of columns in DSN_QUERYINFO_TABLE (continued)

Column name	Data type	Description
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
SEQNO	INTEGER NOT NULL WITH DEFAULT	The sequence number for this row if QI_DATA exceeds the size of its column.

Table 194. Descriptions of columns in DSN_QUERYINFO_TABLE (continued)

Column name	Data type	Description
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
TYPE	CHAR(8) NOT NULL WITH DEFAULT	The type of the output for this row: **A** This row is for a statement that DB2 attempts to run on an accelerator server. The value in column REASON_CODE indicates the outcome.
QI_DATA	CLOB(2M) NOT NULL WITH DEFAULT	When TYPE='A': • For REASON_CODE values other than 0, this value is the description of the REASON_CODE value. • For a REASON_CODE value of 0, this value is the statement text, after it is converted for processing by the accelerator.
SERVICE_INFO	BLOB(2M) NOT NULL WITH DEFAULT	IBM internal use only.
QB_INFO_ROWID	ROWID NOT NULL GENERATED ALWAYS	IBM internal use only.

Notes:

1. The REASON_CODE column has the following values:

   0 The query block qualifies for routing to an accelerator server. The values of QINAME1 and QINAME2 identify the accelerator server.

   For example, for version 1 of the IBM DB2 Analytics Accelerator for z/OS, the associated data mart name is recorded in the QINAME2 column, with the following naming convention: `data-mart-name@accelerator-name@digits`.

   1 No active accelerator server was found when EXPLAIN was executed.

   2

   Special register CURRENT QUERY ACCELERATION is set to NONE.

   3 DB2 classified the query as a short-running query, or DB2 determined that sending the query to an accelerator server provided no performance advantage.

   4 The query is not read-only.

   6 The cursor is defined as a scrollable cursor or rowset cursor.

   7 The query references objects with multiple encoding schemes.

   8 The FROM clause of the query specifies a data change table reference.

9 The query contains a table expression with one or more correlated references to other tables in the same FROM clause.

10 The query contains a reference to a recursive common table expression.

11 The query contains an unsupported expression. The text of the expression is in QI_DATA.

12 The query references a table that meets one of the following conditions:

- The table is not defined in the accelerator server.
- The table is defined in the accelerator server, but is not enabled for processing by an accelerator.

13 The accelerator server that contains the tables that are referenced by the query is not started.

14 A column that is referenced in the query was altered by DB2 after the data was loaded in the accelerator server.

15 The query uses functionality that is available only in DB2 for z/OS Version 10 new-function mode or later, and the functionality is not supported by the accelerator server.

17 The query is an INSERT statement, but the DB2 subsystem parameter DSN6SPRM.QUERY_ACCEL_OPTIONS does not specify option 2 to enable its acceleration.

19 The accelerator server is not at the correct level and does not support a function in the SQL statement. The QI_DATA column contains the function text or expression text that is using the unsupported function for the accelerator server.

20 The rowset cursor is declared WITH RETURN or runs remotely or under an SQL PL routine.

21 The query contains a correlated subquery that is not supported for acceleration.

900-999
For IBM internal use only.

PSPI

Related reference:

Support Portal: IBM DB2 Analytics Accelerator for z/OS

DSN_QUERY_TABLE

The query table, DSN_QUERY_TABLE, contains information about a SQL statement, and displays the statement before and after query transformation.

PSPI

Unlike other EXPLAIN tables, rows in DSN_QUERY_TABLE are not populated for static SQL statements at BIND or REBIND with the EXPLAIN(YES) option.

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
: One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
: You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_QUERY_TABLE.

Table 195. DSN_QUERY_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
TYPE	CHAR(8) NOT NULL	The type of the data in the NODE_DATA column.

Table 195. DSN_QUERY_TABLE description (continued)

Column name	Data type	Description
QUERY STAGE	CHAR(8) NOT NULL WITH DEFAULT	The stage during query transformation when this row is populated.
SEQNO	NOT NULL	The sequence number for this row if NODE_DATA exceeds the size of its column.
NODE_DATA	CLOB(2M)	The XML data containing the SQL statement and its query block, table, and column information.
EXPLAIN_TIME	TIMESTAMP	The EXPLAIN timestamp.
QUERY_ROWID	ROWID NOT NULL GENERATED ALWAYS	The ROWID of the statement.
GROUP MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
HASHKEY	INTEGER NOT NULL	The hash value of the contents in NODE_DATA
HAS_PRED	CHAR(1) NOT NULL	When NODE_DATA contains an SQL statement, this column indicates if the statement contains a parameter marker literal, non-parameter marker literal, or no predicates.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.

Table 195. DSN_QUERY_TABLE description (continued)

Column name	Data type	Description
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

DSN_SORTKEY_TABLE

The sort key table, DSN_SORTKEY_TABLE, contains information about sort keys for all of the sorts required by a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by

user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_SORTKEY_TABLE.

Table 196. DSN_SORTKEY_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.

Table 196. DSN_SORTKEY_TABLE description (continued)

Column name	Data type	Description
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
SORTNO	SMALLINT NOT NULL	The sequence number of the sort
ORDERNO	SMALLINT NOT NULL	The sequence number of the sort key
EXPTYPE	CHAR(3) NOT NULL	The type of the sort key. The possible values are: • 'COL' • 'EXP' • 'QRY'
TEXT	VARCHAR(128) NOT NULL	The sort key text, can be a column name, an expression, or a scalar subquery, or 'Record ID'.
TABNO	SMALLINT NOT NULL	The table number, a number which uniquely identifies the corresponding table reference within a query.
COLNO	SMALLINT NOT NULL	The column number, a number which uniquely identifies the corresponding column within a query. Only applicable when the sort key is a column.

Table 196. DSN_SORTKEY_TABLE description (continued)

Column name	Data type	Description
DATATYPE	CHAR(18)	The data type of sort key. The possible values are • 'HEXADECIMAL' • 'CHARACTER' • 'PACKED FIELD ' • 'FIXED(31)' • 'FIXED(15)' • 'DATE' • 'TIME' • 'VARCHAR' • 'PACKED FLD' • 'FLOAT' • 'TIMESTAMP' • 'UNKNOWN DATA TYPE'
LENGTH	INTEGER NOT NULL	The length of sort key.
CCSID	INTEGER NOT NULL	IBM internal use only.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

DSN_SORT_TABLE

The sort table, DSN_SORT_TABLE, contains information about the sort operations required by a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'

One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'

You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_SORT_TABLE.

Table 197. DSN_SORT_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.

Table 197. DSN_SORT_TABLE description (continued)

Column name	Data type	Description
COLLID	VARCHAR(128) NOT NULL	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
SORTC	CHAR(5) NOT NULL WITH DEFAULT	Indicates the reasons for sort of the composite table. The reasons are shown as a series of bytes: • Byte 1 is 'G' if the reason is GROUP BY, or otherwise blank. • The second byte is 'J' if the reason is JOIN, or otherwise blank. • Byte is 'O' if the reason is ORDER BY, or otherwise blank. • The fourth by is 'U' if the reason is uniqueness, or otherwise blank.
SORTN	CHAR(5) NOT NULL WITH DEFAULT	Indicates the reasons for sort of the new table. The reasons are shown as a series of bytes: • The first byte is 'G' if the reason is GROUP BY, or otherwise blank. • The second byte is 'J' if the reason is JOIN, or otherwise blank. • The third byte is 'O' if the reason is ORDER BY, or otherwise blank. • The fourth by is 'U' if the reason is uniqueness, or otherwise blank.
SORTNO	SMALLINT NOT NULL	The sequence number of the sort.
KEYSIZE	SMALLINT NOT NULL	The sum of the lengths of the sort keys.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.

Table 197. DSN_SORT_TABLE description (continued)

Column name	Data type	Description
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

DSN_STATEMENT_CACHE_TABLE

The statement cache table, DSN_STATEMENT_CACHE_TABLE, contains information about the SQL statements in the statement cache, information captured as the results of an EXPLAIN STATEMENT CACHE ALL statement.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

userID You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Unlike other EXPLAIN tables, no instance of DSN_STATEMENT_CACHE_TABLE is created under the SYSIBM qualifier.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table shows the descriptions of the columns in DSN_STATEMENT_CACHE_TABLE.

Table 198. Descriptions of columns in DSN_STATEMENT_CACHE_TABLE

Column name	Data Type	Description
STMT_ID	INTEGER NOT NULL	The statement ID; this value is the EDM unique token for the statement.
STMT_TOKEN	VARCHAR(240)	The statement token; you provide this value as an identification string.
COLLID	VARCHAR(128) NOT NULL	The collection ID: **DSNDYNAMICSQLCACHE** The row originates from the dynamic statement cache **DSNEXPLAINMODEYES** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **DSNEXPLAINMODEEXPLAIN** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register.
PROGRAM_NAME	VARCHAR(128) NOT NULL	The name of the package that performed the initial PREPARE for the statement.
INV_DROPALT	CHAR(1) NOT NULL	This column is not used.
INV_REVOKE	CHAR(1) NOT NULL	This column is not used.
INV_LRU	CHAR(1) NOT NULL	This column is not used.
INV_RUNSTATS	CHAR(1) NOT NULL	This column is not used.
CACHED_TS	TIMESTAMP NOT NULL	The timestamp when the statement was stored in the dynamic statement cache.3 on page 983
USERS	INTEGER NOT NULL	The number of current users of the statement. This number indicates the users that have prepared or run the statement during their current unit of work. 1 on page 982,3 on page 983
COPIES	INTEGER NOT NULL	The number of copies of the statement that are owned by all threads in the system. 1 on page 982,3 on page 983
LINES	INTEGER NOT NULL	The precompiler line number from the initial PREPARE of the statement. 1 on page 982
PRIMAUTH	VARCHAR(128) NOT NULL	The primary authorization ID that did the initial PREPARE of the statement.
CURSQLID	VARCHAR(128) NOT NULL	The CURRENT SQLID that did the initial PREPARE of the statement.
BIND_QUALIFIER	VARCHAR(128) NOT NULL	The BIND qualifier. For unqualified table names, this is the object qualifier.

Table 198. Descriptions of columns in DSN_STATEMENT_CACHE_TABLE (continued)

Column name	Data Type	Description
BIND_ISO	CHAR(2) NOT NULL	The value of the ISOLATION BIND option that is in effect for this statement. The value will be one of the following values: **'UR'** Uncommitted read **'CS'** Cursor stability **'RS'** Read stability **'RR'** Repeatable read
BIND_CDATA	CHAR(1) NOT NULL	The value of the CURRENTDATA BIND option that is in effect for this statement. The value will be one of the following values: **'Y'** CURRENTDATA(YES) **'N'** CURRENTDATA(NO)
BIND_DYNRL	CHAR(1) NOT NULL	The value of the DYNAMICRULES BIND option that is in effect for this statement. The value will be one of the following values: **'B'** DYNAMICRULE(BIND) **'R'** DYNAMICRULES(RUN)
BIND_DEGRE	CHAR(1) NOT NULL	The value of the CURRENT DEGREE special register that is in effect for this statement. The value will be one of the following values: **'A'** CURRENT DEGREE = ANY **'1'** CURRENT DEGREE = 1
BIND_SQLRL	CHAR(1) NOT NULL	The value of the CURRENT RULES special register that is in effect for this statement. The value will be one of the following values: **'D'** CURRENT RULES = DB2 **'S'** CURRENT RULES = SQL
BIND_CHOLD	CHAR(1) NOT NULL	The value of the WITH HOLD attribute of the PREPARE for this statement. The value will be one of the following values: **'Y'** Initial PREPARE specified WITH HOLD **'N'** Initial PREPARE specified WITHOUT HOLD
STAT_TS	TIMESTAMP NOT NULL	Timestamp of the statistics. This is the timestamp when IFCID 318 is started. 2 on page 982
STAT_EXEC	INTEGER NOT NULL	This column is deprecated. Use STAT_EXECB instead.
STAT_GPAG	INTEGER NOT NULL	This column is deprecated. Use STAT_GPAGB instead. 1 on page 982
STAT_SYNR	INTEGER NOT NULL	This column is deprecated. Use STAT_SYNRB instead. 1 on page 982
STAT_WRIT	INTEGER NOT NULL	This column is deprecated. Use STAT_WRITB instead. 1 on page 982
STAT_EROW	INTEGER NOT NULL	This column is deprecated. Use STAT_EROWB instead. 1 on page 982
STAT_PROW	INTEGER NOT NULL	This column is deprecated. Use STAT_PROWB instead. 1 on page 982
STAT_SORT	INTEGER NOT NULL	This column is deprecated. Use STAT_SORTB instead. 1 on page 982
STAT_INDX	INTEGER NOT NULL	This column is deprecated. Use STAT_SORTB instead.

Table 198. Descriptions of columns in DSN_STATEMENT_CACHE_TABLE (continued)

Column name	Data Type	Description
STAT_RSCN	INTEGER NOT NULL	This column is deprecated. Use STAT_SORTB instead.
STAT_PGRP	INTEGER NOT NULL	This column is deprecated. Use STAT_SORTB instead.
STAT_ELAP	FLOAT NOT NULL	The accumulated elapsed time that is used for the statement. 2 on page 982
STAT_CPU	FLOAT NOT NULL	The accumulated CPU time that is used for the statement. 2 on page 982
STAT_SUS_SYNIO	FLOAT NOT NULL	The accumulated wait time for synchronous I/O operations for the statement. 2 on page 982
STAT_SUS_LOCK	FLOAT NOT NULL	The accumulated wait time for lock requests for the statement. 2 on page 982
STAT_SUS_SWIT	FLOAT NOT NULL	The accumulated wait time for synchronous execution unit switch for the statement. 2 on page 982
STAT_SUS_GLCK	FLOAT NOT NULL	The accumulated wait time for global locks for this statement. 2 on page 982
STAT_SUS_OTHR	FLOAT NOT NULL	The accumulated wait time for read activity that is done by another thread. 2 on page 982
STAT_SUS_OTHW	FLOAT NOT NULL	The accumulated wait time for write activity done by another thread. 2 on page 982
STAT_RIDLIMT	INTEGER NOT NULL	This column is deprecated. Use STAT_SORTB instead.
STAT_RIDSTOR	INTEGER NOT NULL	This column is deprecated. Use STAT_SORTB instead.
EXPLAIN_TS	TIMESTAMP NOT NULL	The timestamp for when the statement cache table is populated.
SCHEMA	VARCHAR(128) NOT NULL	The value of the CURRENT SCHEMA special register.
STMT_TEXT	CLOB(2M) NOT NULL	The statement that is being explained.
STMT_ROWID	ROWID NOT NULL GENERATED ALWAYS	The ROWID of the statement.
BIND_RO_TYPE	CHAR(1) NOT NULL WITH DEFAULT	The current specification of the REOPT option for the statement3 on page 983: **'N'** REOPT(NONE) or its equivalent **'1'** REOPT(ONCE) or its equivalent **'A'** REOPT(AUTO) or its equivalent **'O'** The current plan is deemed optimal and there is no need for REOPT(AUTO)
BIND_RA_TOT	INTEGER NOT NULL WITH DEFAULT	The total number of REBIND commands that have been issued for the dynamic statement because of the REOPT(AUTO) option.1 on page 982,3 on page 983
GROUP_MEMBER	VARCHAR(24) NOT NULL WITH DEFAULT	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
STAT_EXECB	BIGINT NOT NULL WITH DEFAULT	The number of times this statement has been run. For a statement with a cursor, this is the number of OPENs.2 on page 982

Table 198. Descriptions of columns in DSN_STATEMENT_CACHE_TABLE (continued)

Column name	Data Type	Description
STAT_GPAGB	BIGINT NOT NULL WITH DEFAULT	The number of getpage operations that are performed for the statement. 2 on page 982
STAT_SYNRB	BIGINT NOT NULL WITH DEFAULT	The number of synchronous buffer reads that are performed for the statement. 2 on page 982
STAT_WRITB	BIGINT NOT NULL WITH DEFAULT	The number of buffer write operations that are performed for the statement. 2 on page 982
STAT_EROWB	BIGINT NOT NULL WITH DEFAULT	The number of rows that are examined for the statement. 2 on page 982
STAT_PROWB	BIGINT NOT NULL WITH DEFAULT	The number of rows that are processed for the statement. 2 on page 982
STAT_SORTB	BIGINT NOT NULL WITH DEFAULT	The number of sorts that are performed for the statement.2 on page 982
STAT_INDXB	BIGINT NOT NULL WITH DEFAULT	The number of index scans that are performed for the statement.2 on page 982
STAT_RSCNB	BIGINT NOT NULL WITH DEFAULT	The number of table space scans that are performed for the statement.2 on page 982
STAT_PGRPB	BIGINT NOT NULL WITH DEFAULT	The number of parallel groups that are created for the statement.2 on page 982
STAT_RIDLIMTB	BIGINT NOT NULL WITH DEFAULT	The number of times a RID list was not used because the number of RIDs would have exceeded DB2 limits.2 on page 982
STAT_RIDSTORB	BIGINT NOT NULL WITH DEFAULT	The number of times a RID list was not used because there is not enough storage available to hold the list of RIDs.2 on page 982
LITERAL_REPL	CHAR(1) NOT NULL WITH DEFAULT	Identifies cached statements where the literal values are replaced by the '&' symbol:3 on page 983 **'R'** The statement is prepared with CONCENTRATE STATEMENTS WITH LITERALS behavior and the literal constants in the statement have been replaced with '&' . **'D'** This statement is a duplicate statement instance with different literal reusability criteria. **blank** Literal values are not replaced.
STAT_SUS_LATCH	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for latch requests for the statement.
STAT_SUS_PLATCH	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for page latch requests for the statement.
STAT_SUS_DRAIN	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for drain lock requests for the statement.
STAT_SUS_CLAIM	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for claim count requests for the statement.
STAT_SUS_LOG	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for log writer requests for the statement.
EXPANSION_REASON	CHAR(2) NOT NULL WITH DEFAULT	This column applies only to statements that reference archive tables or temporal tables. For other statements, this column is blank, which means that the query does not contain query transformation.

Table 198. Descriptions of columns in DSN_STATEMENT_CACHE_TABLE (continued)

Column name	Data Type	Description
ACCELERATED	CHAR(10)	Identifies whether a cached dynamic statement was prepared for acceleration to an accelerator server. Possible values are: **'NO'** The cached statement was not prepared for acceleration. This is the default value. This value also applies to cached statements under the following conditions: • The query acceleration behavior was not specified or was explicitly set to NONE when the dynamic statement was prepared. • A query acceleration behavior other than ALL was specified when the dynamic statement was prepared. DB2 did not prepare the statement for acceleration because it did not qualify for acceleration based on the query acceleration behavior that was specified. **'YES'** The cached statement was prepared for acceleration to an accelerator server based on the query acceleration behavior that was specified. If query acceleration behavior is specified when a dynamic statement is prepared, DB2 can consider this cache entry for a possible cache match during the prepare operation of the dynamic statement. If query acceleration behavior is not specified, or is explicitly set to NONE when the statement is prepared, DB2 does not consider this cache entry for a cache match during the prepare operation. **'NEVER'** The cached statement was not prepared for acceleration to an accelerated server, because the statement can never be accelerated. • If the query acceleration behavior is set to ENABLE, ENABLE WITH FAILBACK, or ELIGIBLE for the prepare of the statement, DB2 considers this cache entry first as a possible cache match during the prepare operation. This action verifies whether the statement was cached previously as one that can never be accelerated. • If the query acceleration behavior is set to ALL for the prepare of the statement, DB2 does not consider this cache entry as a possible cache match during the prepare operation. • If the query acceleration behavior is not specified, or is explicitly set to NONE for the prepare of the statement, DB2 does not consider this cache entry as a possible cache match during the prepare operation.

Table 198. Descriptions of columns in DSN_STATEMENT_CACHE_TABLE (continued)

Column name	Data Type	Description
ACCELERATED (continued)		**'ACCEL-ONLY'** The cached statement was prepared for acceleration to an accelerator. The statement references at least one accelerator-only table and can only be prepared to execute in the accelerator server. If query acceleration behavior is specified when a dynamic statement is prepared, DB2 can consider this cache entry for a possible cache match during the prepare operation of the dynamic statement. If query acceleration behavior is not specified, or is explicitly set to NONE when the statement is prepared, DB2 does not consider this cache entry for a cache match during the prepare operation. Query acceleration behavior is specified by either the QUERY_ACCELERATION subsystem parameter, the QUERYACCELERATION bind option, or the CURRENT QUERY ACCELERATION special register, and depends on their order of precedence. The order of precedence (lowest to highest) is: • The QUERY_ACCELERATION subsystem parameter • The QUERYACCELERATION bind option, if specified • An explicit SET CURRENT QUERY ACCELERATION statement
STAT_ACC_ELAP	BIGINT	The accumulated elapsed time for the accelerator.
STAT_ACC_CPU	BIGINT	The accumulated CPU time for the accelerator.
STAT_ACC_ROW	BIGINT	The accumulated number of rows that are returned from the accelerator.
STAT_ACC_BYTE	BIGINT	The accumulated number of bytes that are returned from the accelerator.
STAT_ACC_1ROW	BIGINT	The time waited for the first row of the query result to be available from the accelerator.
STAT_ACC_DB2	BIGINT	The total time the accelerator waited for DB2 to request query results.
STAT_ACC_EXEC	BIGINT	The accumulated execution time for the accelerator.
STAT_ACC_WAIT	BIGINT	The accumulated queue wait time for the accelerator.
ACCEL_OFFLOAD_ELIGIBLE	CHAR(1)	**'NO'** The statement is not eligible for acceleration. This is the default value. **'YES'** The statement is a candidate for acceleration when an accelerator server is available to the DB2 subsystem.
ACCELERATOR_NAME	VARCHAR(128)	The concatenated name of the accelerator server that processed the query.

Notes:

1. If the specified value exceeds 2147483647, the column contains the value 2147483647.
2. Statistics are cumulative, across executions of the same statement, and across threads, if the value of COLLID is DSNDYNAMICSQLCACHE. If the value of

COLLID is DSNEXPLAINMODEYES, the values are for a single run of the statement only. If the value of COLLID is DSNEXPLAINMODE EXPLAIN, the values of all statistics columns are 0.

3. The column is not applicable when the value of the COLLID column is 'DSNEXPLAINMODEYES' or 'DSNEXPLAINMODEEXPLAIN'

PSPI

DSN_STATEMNT_TABLE

The statement table, DSN_STATEMNT_TABLE, contains information about the estimated cost of specified SQL statements.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the content of each column in STATEMNT_TABLE.

Table 199. Descriptions of columns in DSN_STATEMNT_TABLE

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL WITH DEFAULT	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.

Table 199. Descriptions of columns in DSN_STATEMNT_TABLE (continued)

Column name	Data type	Description
GROUP_MEMBER	VARCHAR(24) NOT NULL WITH DEFAULT	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
EXPLAIN_TIME	TIMESTAMP NOT NULL WITH DEFAULT	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
STMT_TYPE	CHAR(6) NOT NULL WITH DEFAULT	The type of statement being explained. Possible values are: **SELECT** SELECT **INSERT** INSERT **UPDATE** UPDATE **MERGE** MERGE **DELETE** DELETE **TRUNCA** TRUNCATE **SELUPD** SELECT with FOR UPDATE OF **DELCUR** DELETE WHERE CURRENT OF CURSOR **UPDCUR** UPDATE WHERE CURRENT OF CURSOR
COST_CATEGORY	CHAR(1) NOT NULL WITH DEFAULT	Indicates if DB2 was forced to use default values when making its estimates. Possible values: **A** Indicates that DB2 had enough information to make a cost estimate without using default values. **B** Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A.
PROCMS	INTEGER NOT NULL WITH DEFAULT	The estimated processor cost, in milliseconds, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported. If an accelerator is used, the difference is reflected in this value.

Table 199. Descriptions of columns in DSN_STATEMNT_TABLE (continued)

Column name	Data type	Description
PROCSU	INTEGER NOT NULL WITH DEFAULT	The estimated processor cost, in service units, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 service units. If the estimated value exceeds this maximum, the maximum value is reported. If an accelerator is used, this value represents the estimated cost including any impact of acceleration.
REASON	VARCHAR(254) WITH DEFAULT	A string that indicates the reasons for putting an estimate into cost category B. **ACCELMODEL ELIGIBLE** The query is eligible for acceleration. **ACCELMODEL NOT ELIGIBLE** The query is not eligible for acceleration. **HAVING CLAUSE** A subselect in the SQL statement contains a HAVING clause. **HOST VARIABLES** The statement uses host variables, parameter markers, or special registers. **OPTIMIZATION HINTS** An statement-level access path, or PLAN_TABLE access path hint is applied to the statement, or APREUSE(ERROR/WARN) is applied for the package. **PROFILEID** ***value*** When profile monitoring is used for the statement, the value of the PROFILEID column in SYSIBM.DSN_PROFILE_TABLE. **REFERENTIAL CONSTRAINTS** Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement. **TABLE CARDINALITY** The cardinality statistics are missing for one or more of the tables that are used in the statement, or the statement used materialized views or table expressions. **TRIGGERS** Triggers are defined on the target table of an insert, update, or delete operation. **UDF** The statement uses user-defined functions.
STMT_ENCODE	CHAR(1) WITH DEFAULT	Encoding scheme of the statement. If the statement represents a single CCSID set, the possible values are: **A** ASCII **E** EBCDIC **U** Unicode If the statement has multiple CCSID sets, the value is M.
TOTAL_COST	FLOAT NOT NULL WITH DEFAULT	The overall estimated cost of the statement. If an accelerator is used, the benefit is reflected in this value.. Use this value for reference purposes only. DB2 does not always choose the access path that has the lowest TOTAL_COST value. DB2 also uses other factors during access path selection, such as the reliability of the filter factor estimates.

Table 199. Descriptions of columns in DSN_STATEMNT_TABLE (continued)

Column name	Data type	Description
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

Related reference:

Support Portal: IBM DB2 Analytics Accelerator for z/OS

DSN_STRUCT_TABLE

The structure table, DSN_STRUCT_TABLE, contains information about all of the query blocks in a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_STRUCT_TABLE

Table 200. DSN_STRUCT_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.
QBLOCKNO	SMALLINT NOT NULL	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
APPLNAME	VARCHAR(24) NOT NULL	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
PARENT	SMALLINT NOT NULL	The parent query block number of the current query block in the structure of SQL text; this is the same as the PARENT_QBLOCKNO in PLAN_TABLE.
TIMES	FLOAT NOT NULL	The estimated number of rows returned by Data Manager; also the estimated number of times this query block is executed.
ROWCOUNT	INTEGER NOT NULL	The estimated number of rows returned by RDS (Query Cardinality).

Table 200. DSN_STRUCT_TABLE description (continued)

Column name	Data type	Description
ATOPEN	CHAR(1) NOT NULL	Whether the query block is moved up for do-at-open processing; 'Y' if done-at-open; 'N': otherwise.
CONTEXT	CHAR(10) NOT NULL	This column indicates what the context of the current query block is. The possible values are: • 'TOP LEVEL' • 'UNION' • 'UNION ALL' • 'PREDICATE' • 'TABLE EXP' • 'UNKNOWN'
ORDERNO	SMALLINT NOT NULL	Not currently used.
DOATOPEN_PARENT	SMALLINT NOT NULL	The parent query block number of the current query block; Do-at-open parent if the query block is done-at-open, this may be different from the PARENT_QBLOCKNO in PLAN_TABLE.
QBLOCK_TYPE	CHAR(6) NOT NULL WITH DEFAULT	This column indicates the type of the current query block. The possible values are • 'SELECT' • 'INSERT' • 'UPDATE' • 'DELETE' • 'SELUPD' • 'DELCUR' • ''UPDCUR' • 'CORSUB' • 'NCOSUB' • 'TABLEX' • 'TRIGGR' • 'UNION' • 'UNIONA' • 'CTE' It is equivalent to QBLOCK_TYPE column in PLAN_TABLE, except for CTE.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
QUERY_STAGE	CHAR(8) NOT NULL	IBM internal use only.

Table 200. DSN_STRUCT_TABLE description (continued)

Column name	Data type	Description
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
ORIGIN	CHAR(1) NOT NULL WITH DEFAULT	Indicates the origin of the query block: **Blank** Generated by DB2 **C** Column mask **R** Row permission **U** Specified by the user
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.

PSPI

DSN_VIEWREF_TABLE

The view reference table, DSN_VIEWREF_TABLE, contains information about all of the views and materialized query tables that are used to process a query.

PSPI

Recommendation: Do not manually insert data into system-maintained EXPLAIN tables, and use care when deleting obsolete EXPLAIN table data. The data is intended to be manipulated only by the DB2 EXPLAIN function and optimization tools. Certain optimization tools depend on instances of the various EXPLAIN tables. Be careful not to delete data from or drop instances EXPLAIN tables that are created for these tools.

Qualifiers

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

Sample CREATE TABLE statement

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the *prefix*.SDSNSAMP library.

Column descriptions

The following table describes the columns of DSN_VIEWREF_TABLE.

Table 201. DSN_VIEWREF_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL WITH DEFAULT	A number that identifies the statement that is being explained. The origin of the value depends on the context of the row: **For rows produced by EXPLAIN statements** The number specified in the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. **For rows not produced by EXPLAIN statements** DB2 assigns a number that is based on the line number of the SQL statement in the source program. When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in certain rare cases, the value is not guaranteed to be unique. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, if the QUERYNO clause is specified, its value is used by DB2. Otherwise DB2 assigns a number based on the line number of the SQL statement in the compiled SQL function or native SQL procedure.

Table 201. DSN_VIEWREF_TABLE description (continued)

Column name	Data type	Description
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column is not used, and is blank.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the specific name of the compiled SQL function or native SQL procedure.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the version identifier of the compiled SQL function or native SQL procedure.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID: **'DSNDYNAMICSQLCACHE'** The row originates from the dynamic statement cache **'DSNEXPLAINMODEYES'** The row originates from an application that specifies YES for the value of the CURRENT EXPLAIN MODE special register. **'DSNEXPLAINMODEEXPLAIN'** The row originates from an application that specifies EXPLAIN for the value of the CURRENT EXPLAIN MODE special register. When the SQL statement is embedded in a compiled SQL function or native SQL procedure, this column indicates the schema name of the compiled SQL function or native SQL procedure.
CREATOR	VARCHAR(128) NOT NULL WITH DEFAULT	Authorization ID of the owner of the object.
NAME	VARCHAR(128)	Name of the object.
TYPE	CHAR(1) NOT NULL WITH DEFAULT	The type of the object: **'V'** View **'R'** MQT that has been used to replace the base table for rewrite **'M'** MQT
MQTUSE	SMALLINT WITH DEFAULT	IBM internal use only.

Table 201. DSN_VIEWREF_TABLE description (continued)

Column name	Data type	Description
EXPLAIN_TIME	TIMESTAMP NOT NULL WITH DEFAULT	The time when the EXPLAIN information was captured: **All cached statements** When the statement entered the cache, in the form of a full-precision timestamp value. **Non-cached static statements** When the statement was bound, in the form of a full precision timestamp value. **Non-cached dynamic statements** When EXPLAIN was executed, in the form of a value equivalent to a CHAR(16) representation of the time appended by 4 zeros.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
SECTNOI	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. The value is taken from the same column in SYSPACKSTMT or SYSSTMT tables and can be used to join tables to reconstruct the access path for the statement. This column is applicable only for static statements. The default value of -1 indicates EXPLAIN information that was captured in DB2 9 or earlier.

PSPI

Input tables

DB2-supplied user input tables enable you to provide certain types of information, such as performance tuning information, to DB2.

DSN_VIRTUAL_INDEXES

The virtual indexes table, DSN_VIRTUAL_INDEXES, enables optimization tools to test the effect of creating and dropping indexes on the performance of particular queries.

PSPI

Your subsystem or data sharing group can contain more than one of these tables:

'SYSIBM'
One instance of this table can be created with the SYSIBM qualifier. DB2 and SQL optimization tools might use the table and the data that it contains. The table is created when you run job DSNTIJSG when you install or migrate DB2.

'*user-ID*'
You can create additional instances of EXPLAIN tables that are qualified by user ID. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind. They are also populated when you specify EXPLAIN(YES) or EXPLAIN(ONLY) in a BIND or REBIND command. SQL optimization tools might also create EXPLAIN tables that

are qualified by a user ID. You can find the SQL statement for creating an instance of these tables in member DSNTESC of the SDSNSAMP library.

PSPI

If virtual indexes are being used, queries that are submitted using EXPLAIN PLAN FOR *explainable-sql-statement* will use virtual indexes as well as regular indexes during optimization. Any virtual index specifications in the DSN_VIRTUAL_INDEXES table will be used during query optimization. The following table describes the columns in DSN_VIRTUAL_INDEXES.

Recommendation: Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

Table 202. DSN_VIRTUAL_INDEXES description

Column name	Data type	Description
TBCREATOR	VARCHAR(128)	The schema or authorization ID of the owner of the table on which the index is being created or dropped.
TBNAME	VARCHAR(128)	The name of the table on which the index is being created or dropped.
IXCREATOR	VARCHAR(128)	The schema or authorization ID of the owner of the index.
IXNAME	VARCHAR(128)	The index name.
ENABLE	CHAR(1)	Indicates whether this index should be considered in the scenario that is being tested. This column can have one of the following values: **Y** Use this index. **N** Do not use this index. If this column contains 'Y', but the index definition is not valid, the index is ignored.
MODE	CHAR(1)	Indicates whether the index is being created or dropped. This column can have one of the following values: **C** This index is to be created. **D** This index is to be dropped.
UNIQUERULE	CHAR(1)	Indicates whether the index is unique. This column can have one of the following values: **D** The index is not unique. (Duplicates are allowed.) **U** This index is unique.
COLCOUNT	SMALLINT	The number of columns in the key.
CLUSTERING	CHAR(1)	Indicates whether the index is clustered. This column can have one of the following values: **Y** The index is clustered. **N** The index is not clustered.
NLEAF	INTEGER	The number of active leaf pages in the index. If unknown, the value is -1.
NLEVELS	SMALLINT	The number of levels in the index tree. If unknown, the value is -1.

Table 202. DSN_VIRTUAL_INDEXES description (continued)

Column name	Data type	Description
INDEXTYPE	CHAR(1)	The index type. This column can have one of the following values: **2** The index is a nonpartitioned secondary index. **D** The index is a data-partitioned secondary index.
PGSIZE	SMALLINT	The size, in kilobytes, of the leaf pages in the index. This column can have one of the following values: 4, 8, 16, or 32.
FIRSTKEYCARDF	FLOAT	The number of distinct values of the first key column. If unknown, the value is -1.
FULLKEYCARDF	FLOAT	The number of distinct values of the key. If unknown, the value is -1.
CLUSTERRATIOF	FLOAT	The percentage of rows that are in clustering order. Multiply this column value by 100 to get the percent value. For example, a value of .9125 in this column indicates that 91.25%. of the rows are in clustering order. If unknown, the value is -1.
PADDED	CHAR(1)	Indicates whether keys within the index are padded for varying-length column data. This column can have one of the following values: **Y** The keys are padded. **N** The keys are not padded.
COLNO1	SMALLINT	The column number of the first column in the index key.
ORDERING1	CHAR(1)	Indicates the order of the first column in the index key. This column can have one of the following values: **A** Ascending **D** Descending
COLNO*n*	SMALLINT	The column number of the *n*th column in the index key, where *n* is a number between 2 and 64, including 2 and 64. If the number of index keys is less than *n*, this column is null.
ORDERING*n*	CHAR(1)	Indicates the order of the *n*th column in the index key, where *n* is a number between 2 and 64, including 2 and 64. This column can have one of the following values: **A** Ascending **D** Descending If the number of index keys is less than *n*, this column is null.

Related tasks:

Analyzing index impact (DB2 Query Workload Tuner for z/OS)

Related reference:

DB2 Query Workload Tuner for z/OS

DSN_USERQUERY_TABLE

The DSN_USERQUERY_TABLE table identifies statements whose access paths are influenced. The values identify the statements and the method that is used to influence access path selection. Values in the DSN_USERQUERY_TABLE are used to populate certain catalog tables when a BIND QUERY command is issued.

PSPI

DSN_USERQUERY_TABLE is created when you modify and run the DSNTESH sample job.

Create table statement

The following statement creates a user query table:

```
 CREATE TABLE userid.DSN_USERQUERY_TABLE
 (
 QUERYNO        INTEGER      NOT NULL PRIMARY KEY,
 SCHEMA         VARCHAR(128) NOT NULL DEFAULT ' ',
 HINT_SCOPE     SMALLINT     NOT NULL DEFAULT 0,
 QUERY_TEXT     CLOB(2M)     NOT NULL,
 QUERY_ROWID    ROWID        NOT NULL  GENERATED ALWAYS,
 QUERYID        BIGINT       NOT NULL DEFAULT 0,
 USERFILTER     CHAR(8)      NOT NULL DEFAULT ' ',
 OTHER_OPTIONS  CHAR(128)    NOT NULL DEFAULT ' ',
 COLLECTION     VARCHAR(128) NOT NULL DEFAULT ' ',
 PACKAGE        VARCHAR(128) NOT NULL DEFAULT ' ',
 VERSION        VARCHAR(128) NOT NULL DEFAULT ' ',
 REOPT          CHAR(1)      NOT NULL DEFAULT ' ',
 STARJOIN       CHAR(1)      NOT NULL DEFAULT ' ',
 MAX_PAR_DEGREE INTEGER      NOT NULL DEFAULT -1,
 DEF_CURR_DEGREE CHAR(3)     NOT NULL DEFAULT ' ',
 SJTABLES       INTEGER      NOT NULL DEFAULT -1,
 OTHER_PARMS    VARCHAR(128) NOT NULL DEFAULT ' '

 ) IN database-name.table-space-name

CCSID UNICODE;
```

Column descriptions

The following table describes the columns of DSN_USERQUERY_TABLE.

Table 203. DSN_USERQUERY_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL PRIMARY KEY	The unique identifier of the query, used to correlate with PLAN_TABLE rows for statement-level access paths.
SCHEMA	VARCHAR(128) NOT NULL DEFAULT ' '	Default schema name of unqualified database objects, excluding functions, in the query, or blank
HINT_SCOPE	SMALLINT NOT NULL DEFAULT 0	The scope at which matching applies. **0** System-level access path hint or the default value **1** Package-level access plan hint.
QUERY_TXT	CLOB(2M) NOT NULL	The text of the SQL statement.
USERFILTER	CHAR(8) NOT NULL	A filter name that you can specify to group a set of rows together, or blank. This value can be used to delete a set of related rows at the same time with a single FREEQUERY command.
OTHER_OPTIONS	CHAR(128) NOT NULL DEFAULT ' '	For IBM internal use only, or blank

Table 203. DSN_USERQUERY_TABLE description (continued)

Column name	Data type	Description
COLLECTION	VARCHAR(128) NOT NULL DEFAULT ' '	The collection name of the package from the SYSIBM.SYSPACKAGE catalog table. This value is optional when the value of the HINT_SCOPE column is 0.
PACKAGE	VARCHAR(128) NOT NULL DEFAULT ' '	The name of the package for the SYSIBM.SYSPACKAGE catalog table. This value is optional when the value of the HINT_SCOPE column is 0.
VERSION	VARCHAR(128) NOT NULL DEFAULT ' '	The version of the package for retrieval of bind options for the SYSIBM.SYSPACKAGE catalog table, or '*'. This value is optional when the value of the HINT_SCOPE column is 0. When '*' is specified, DB2 uses only COLLECTION and PACKAGE values to look up rows in the SYSIBM.SYSPACKAGE and SYSIBM.SYSQUERY catalog tables.
REOPT	VARCHAR(128) NOT NULL DEFAULT ' '	The value of the REOPT bind option: **'A'** REOPT(AUTO) **'1'** REOPT(ONCE) **'N'** REOPT(NONE) **'Y'** REOPT(ALWAYS) **blank** Not specified.
STARJOIN	CHAR(1) NOT NULL DEFAULT ' '	Whether star join processing was enabled for the query: **'Y'** STARJOIN enabled. **'N'** STARJOIN disabled. **blank** Not specified.
MAX_PAR_DEGREE	INTEGER NOT NULL DEFAULT -1	The maximum degree of parallelism or -1 if unspecified.
DEF_CURR_DEGREE	CHAR(3) NOT NULL DEFAULT ' '	Whether parallelism was enabled: **'ONE'** Parallelism disabled. **'ANY'** Parallelism enabled. **blank** Not specified.
SJTABLES	INTEGER NOT NULL DEFAULT -1	The minimum number of tables to qualify for the star join processing, or -1 when not specified.
QUERYID	BIGINT NOT NULL DEFAULT 0	Identifies relevant access plan hint information in the SYSIBM.SYSQUERY and SYSIBM.SYSQUERYPLAN catalog tables.
OTHER_PARMS	VARCHAR(128) NOT NULL	For IBM internal use only, or BLANK

PSPI

Related tasks:

Influencing access path selection

Related reference:

Tables for influencing access path selection

BIND QUERY (DSN) (DB2 Commands)

Profile tables

You can create and populate profile tables to monitor threads and connections, customize certain subsystem parameters in particular contexts, and stabilize dynamic SQL statements.

PSPI

Experienced administrators can create and populate profile tables to monitor a DB2 subsystem.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

A complete set of profile tables and related indexes includes the following objects:

- SYSIBM.DSN_PROFILE_TABLE
- SYSIBM.DSN_PROFILE_HISTORY
- SYSIBM.DSN_PROFILE_ATTRIBUTES
- SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- SYSIBM.DSN_PROFILE_TABLE_IX_ALL
- SYSIBM.DSN_PROFILE_TABLE_IX2_ALL
- SYSIBM.DSN_PROFILE_ATTRIBUTES_IX_ALL

In addition to the profile tables that are described below, other DB2-supplied user tables can be used in conjunction with monitor profiles, including instances of the following EXPLAIN and run time information tables:

- PLAN_TABLE
- DSN_STATEMNT_TABLE
- DSN_FUNCTION_TABLE

PSPI

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

Creating profiles

Monitoring threads and connections by using profiles

Modeling a production environment on a test subsystem

Optimizing subsystem parameters for SQL statements by using profiles

Related reference:

DSN_FUNCTION_TABLE

DSN_STATEMNT_TABLE

PLAN_TABLE

SYSIBM.DSN_PROFILE_TABLE

Profile tables identify contexts in which DB2 takes particular actions such as resource monitoring, subsystem parameter customization, and dynamic SQL stabilization. The contexts might identify statements, threads, or connections that are based on information about the originating application, system, or user.

PSPI Each row in the profile table, SYSIBM.DSN_PROFILE_TABLE, defines a profile. A *profile* is a set of criteria that identifies a particular context on a DB2 subsystem. Examples include threads, connections, or SQL statements that have particular attributes.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

The following table describes the columns in SYSIBM.DSN_PROFILE_TABLE.

Table 204. SYSIBM.DSN_PROFILE_TABLE description

Column name	Data type	Description
AUTHID	VARCHAR(128)	The authorization ID of a monitored user. For profiles that monitor system resources, this value is optional if a value for ROLE is specified. This value must be null if values are specified for COLLID, PKGNAME, LOCATION, or PRDID. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.
PLANNAME	VARCHAR(24)	The name of a plan. This value must be null if values are specified for AUTHID, LOCATION, ROLE, PKGNAME, or PRDID columns for monitoring system resources.
COLLID	VARCHAR(128)	A collection identifier of a monitored collection. For profiles that monitor system resources, this value is optional if the value of PKGNAME is specified, and this value must be null if values are specified for AUTHID, LOCATION, ROLE, or PRDID. For other types of profiles, this column is required when PKGNAME is specified. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.

Table 204. SYSIBM.DSN_PROFILE_TABLE description (continued)

Column name	Data type	Description
PKGNAME	VARCHAR(128)	A package name of a monitored plan. This value is optional if a value for COLLID is specified. This value must be null if values are specified for AUTHID, LOCATION, ROLE, or PRDID For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.
LOCATION	VARCHAR(254)	One of the following items: • The IPv4 or IPv6 IP address of a remote client. • IP address 127.0.0.1 to specify the local DB2 subsystem. • The domain name of a remote client, for example: `stlmvs1.svl.example.com` • One of the following values that remote clients use to connect to the server: – Database name. – Location name. – Location alias. The value is interpreted as a location name when the name string is less than or equal to 16 bytes and does not contain colon (:) or period (.) characters. When these characters are found, the value is checked for a valid IP address or a valid domain name. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.
PROFILEID	INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY NOT NULL	The unique identifier for the profile that is defined by this row.
PROFILE_TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	The time when the row was inserted or updated.
PROFILE_ENABLED	CHAR(1) NOT NULL WITH DEFAULT 'Y'	Indicates whether the profile is enabled. This column can have one of the following values: **Y** The profile is enabled. **N** The profile is disabled.
GROUP_MEMBER	VARCHAR(24)	The name of the DB2 member in a data sharing group. The column can be blank. When the column is blank, the row applies to a DB2 subsystem that is not part of a data sharing group, or to every DB2 subsystem in a data sharing group.

Table 204. SYSIBM.DSN_PROFILE_TABLE description (continued)

Column name	Data type	Description
REMARKS	VARCHAR(762)	Comments about this profile.
ROLE	VARCHAR(128) WITH DEFAULT NULL	The role of a monitored user or users. This value is optional if the value of AUTHID is specified. This value must be null if values are specified for LOCATION, PKGNAME, or COLLID. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.
PRDID	CHAR(8) WITH DEFAULT NULL	The product-specific identifier of a monitored remote requester. The product identifier is an 8-byte alphanumeric field. The format of product identifier values is *pppvvrrm*, where *ppp* is a 3-letter product code (such as DSN for DB2), *vv* is the version, *rr* is the release, and *m* is the modification level. For example, DSN10015 identifies DB2 10 in new-function mode, the value is 'DSN10015'. The product code (*ppp*) is one of the following values: **AQT** for IBM DB2 Analytics Accelerator for z/OS **ARI** for DB2 Server for VSE & VM **DSN** for DB2 for z/OS **JCC** for IBM Data Server Driver for JDBC and SQLJ **QSQ** for DB2 for i **SQL** for DB2 for Linux, UNIX, and Windows Modification (*m*) values have the following meanings: **0 - 1** Modification levels in conversion and enabling-new-function mode from DB2 Version 8 (CM8, CM8*, ENFM8, and ENFM8*) **2 - 3** Modification levels in conversion and enabling-new-function mode from DB2 9 (CM9, CM9*, ENFM9, and ENFM9*) **4** Not used. **5 - 9** Modification levels in new-function mode. This value must be null if values are specified for AUTHID, PKGNAME, COLLID, AUTHID, or ROLE. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.

Table 204. SYSIBM.DSN_PROFILE_TABLE description (continued)

Column name	Data type	Description
CLIENT_APPLNAME	VARCHAR(255)	The client application name information. It contains the value of the application name or transaction name from the client information that is specified for the connection. This value corresponds to the CURRENT CLIENT_APPLNAME special register. In a distributed environment, if the length of the client application name value exceeds 32 bytes, it is truncated to 32 bytes for filtering criteria. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.
CLIENT_USERID	VARCHAR(255)	The client user ID name information. It contains the value of the client user ID from the client information that is specified for the connection. This value corresponds to the CURRENT CLIENT_USERID special register. If the length of the client user ID name value exceeds 16 bytes, it is truncated to 16 bytes for filtering criteria. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.
CLIENT_ WRKSTNNAME	VARCHAR(255)	The client workstation name information. It contains the value of the workstation name from the client information that is specified for the connection. This value corresponds to the CURRENT CLIENT_WRKSTNNAME special register. If the length of the workstation name value exceeds 18 bytes, it is truncated to 18 bytes for filtering criteria. For profiles that monitor system resources, a single-byte asterisk value ('*') in this column indicates filtering criteria that match all connections or threads that connect to the subsystem. When an asterisk is used, the specified limit applies separately to the number of connections or threads allowed for each requester. Whenever a thread or connection matches a more specific profile, DB2 enforces only the more specific profile.

PSPI

Related concepts:

Profiles for monitoring and controlling DB2 for z/OS subsystems

Related tasks:

Monitoring threads and connections by using profiles

Optimizing subsystem parameters for SQL statements by using profiles

Modeling a production environment on a test subsystem

SYSIBM.DSN_PROFILE_HISTORY

The profile history table, SYSIBM.DSN_PROFILE_HISTORY, contains all of the profiles that were in effect at some point in time.

PSPI

Profiles are defined in the profile table, SYSIBM.DSN_PROFILE_TABLE. Profiles in the profile history table might no longer exist in the profile table. This table is used by optimization tools when diagnosing optimization problems.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

Recommendation: Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

The columns for SYSIBM.DSN_PROFILE_HISTORY are the same as the columns in SYSIBM.DSN_PROFILE_TABLE, except that the REMARKS column is named STATUS instead. The STATUS column has the following possible values:

- 'REJECTED - DUPLICATED SCOPE SPECIFIED'
- 'REJECTED - INVALID LOCATION SPECIFIED'
- 'REJECTED - INVALID SCOPE SPECIFIED'
- 'REJECTED - NO VALID RECORD FOUND IN ATTRIBUTE TABLE'
- 'REJECTED - INVALID SCOPE SPECIFIED. SYSTEM LEVEL MONITORING SCOPE CAN BE SPECIFIED ONLY ON NFM'
- 'REJECTED - INVALID SCOPE SPECIFIED. FOR SYSTEM LEVEL MONITORING, ONLY IP ADDR, PRDID, ROLE AND/OR AUTHID, COLLECTION ID AND/OR PACKAGE NAME CAN BE SPECIFIED'
- 'ACCEPTED - DOMAIN NAME IS RESOLVED INTO IP ADDRESS'
- 'ACCEPTED'

PSPI

SYSIBM.DSN_PROFILE_ATTRIBUTES

The profile attributes table, SYSIBM.DSN_PROFILE_ATTRIBUTES, defines the attributes that are associated with a given profile.

PSPI This table is used by optimization tools when monitoring and tuning statements.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

The following tables describes the columns in SYSIBM.DSN_PROFILE_ATTRIBUTES.

Table 205. SYSIBM.DSN_PROFILE_ATTRIBUTES description

Column name	Data type	Description
PROFILEID	INTEGER NOT NULL FOREIGN KEY REFERENCES SYSIBM.DSN_PROFILE_TABLE ON DELETE CASCADE	The unique identifier for the profile. This value is from the PROFILEID column in SYSIBM.DSN_PROFILE_TABLE.
KEYWORDS	VARCHAR(128) NOT NULL	The function that DB2 performs. The possible function keywords are described below.
ATTRIBUTE1	VARCHAR(1024)	The string attribute that is associated with the function in the KEYWORDS column, if any.
ATTRIBUTE2	INTEGER	The integer attribute that is associated with the function in the KEYWORDS column, if any.
ATTRIBUTE3	FLOAT	The float attribute that is associated with the function in the KEYWORDS column, if any.
ATTRIBUTE_ TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	The time when the row was inserted or updated.
REMARKS	VARCHAR(762)	Comments for this row.

Function keywords for SYSIBM.DSN_PROFILE_ATTRIBUTES

Function keywords specify the action that you want DB2 to perform when the context of an operation, such as a query, thread, or connection, meets the criteria specified in a profile that is defined in the SYSIBM.DSN_PROFILE table. Each keyword uses as many as three attributes that control how the specified action is applied by DB2. These values are specified in the KEYWORDS, ATTRIBUTE1, ATTRIBUTE2, and ATTRIBUTE3 columns of the SYSIBM.DSN_PROFILE_ATTRIBUTES table.

ACCEL_NAME_EXPLAIN

Specifies that a dynamic EXPLAIN statement will evaluate a dynamic SQL query for acceleration to the accelerator that is defined in ATTRIBUTE1. When the CURRENT EXPLAIN MODE special register is set to EXPLAIN, the query is evaluated for acceleration during the prepare phase of a dynamic SQL query.

Attribute1

The name of a real or virtual accelerator.

ACCEL_NAME_EXPLAIN cannot be used for the following queries:

- Static SQL queries
- Dynamic SQL queries that were prepared when using CURRENT EXPLAIN MODE = YES

ACCEL_TABLE_THRESHOLD

Specifies one of the criteria that DB2 uses to determine whether to send a query to an accelerator server. ACCEL_TABLE_THRESHOLD specifies the maximum total table cardinality for a query to be treated as a short running query. If a query has a total table cardinality that is less than ACCEL_TABLE_THRESHOLD, that query is not accelerated.

If you use an accelerator product, contact IBM Support for that accelerator product for information about how to set this keyword.

Attribute1

NULL

Attribute2

A positive integer that represents the total number of rows for a query, or -1. -1 means that this check is disabled. 1000000 is the default.

Attribute3

NULL

ACCEL_RESULTSIZE_THRESHOLD

Specifies one of the criteria that DB2 uses to determine whether to send a query to an accelerator server. ACCEL_RESULTSIZE_THRESHOLD specifies the maximum number of rows that a query that is sent to an accelerator server can return. If a query returns over that number of rows, the query is not sent to an accelerator server.

If you use an accelerator product, contact IBM Support for that accelerator product for information about how to set this keyword.

Attribute1

NULL

Attribute2

A positive integer that represents a number of rows, in thousands, or -1. For example, a value of 20 means 20000 rows. -1 means that result set size checking is disabled. -1 is the default.

Attribute3

NULL

ACCEL_TOTALCOST_THRESHOLD

Specifies one of the criteria that DB2 uses to determine whether to send a query to an accelerator server. ACCEL_TOTALCOST_THRESHOLD specifies the maximum estimated total cost for a query to be treated as a short running query. If a query has a total cost that is less than ACCEL_TOTALCOST_THRESHOLD, that query is not accelerated.

If you use an accelerator product, contact IBM Support for that accelerator product for information about how to set this keyword.

Attribute1

NULL

Attribute2

NULL

Attribute3

A positive float value that represents the estimated total cost for a query, or -1.0. -1.0 means that this check is disabled. 5000 is the default.

BP*name*

Where *name* is any of the following values:

- 0 through 49
- 8K0 through 8K9
- 16K0 through 16K9
- 32K
- 32K1 through 32K9

Attribute1

NULL

Attribute2

A positive integer that specifies the size of the corresponding buffer pool that the test system uses to model the configuration of a production system. When a value of zero or a negative number is specified, the profile attribute entry is rejected.

Attribute3

NULL

MAX_RIDBLOCKS

Specifies the maximum number of RID blocks per RID pool for that a test system uses to model a production system. The attribute value for this keyword corresponds to the value of the MAXRBLK subsystem parameter of the production system. To determine the appropriate value to specify, refer to the value of the PLAN_TABLE.MAX_RIDBLOCKS_EXP column in the EXPLAIN output on the production system.

Attribute1

NULL

Attribute2

0 (zero) or a positive integer that is valid for the value of the MAXRBLK subsystem parameter.

Attribute3

NULL

MIN STAR JOIN TABLES

Specifies that DB2 is to set the minimum number of tables for star join processing (the SJTABLES subsystem parameter) to the value that is specified for the Attribute2 column. This value indicates that DB2 is to use star joins when the number of tables in the statement is equal to or greater than the specified integer.

Attribute1

NULL

Attribute2

An integer between 3 and 225 that specifies the minimum number of tables for star join processing.

Attribute3

NULL

MONITOR CONNECTIONS

Specifies that DB2 monitors the total number of remote connections from application servers, including active connections and live inactive connections.

The connections are filtered only according to the IP address or domain name value that is specified in the LOCATION column of the DSN_PROFILE_TABLE table.

DB2 takes certain actions when the threshold is reached according to the value that is specified in the Attribute1 column. When the total number of connections being queued or suspended reaches the exception threshold, DB2 either issues a message if a WARNING value is specified or starts to fail the connection request with SQLCODE -30041 if EXCEPTION is specified.

Attribute1

Specifies the type and level of detail for messages issued for monitored threads that meet the conditions specified in the profile. One of the following values:

- EXCEPTION
- EXCEPTION_ DIAGLEVEL1
- EXCEPTION_DIAGLEVEL2
- WARNING
- WARNING_DIAGLEVEL1
- WARNING_ DIAGLEVEL2

EXCEPTION values

DB2 aborts the thread, and takes one of the following actions, depending on the Attribute1 value:

EXCEPTION_DIAGLEVEL1 values

DB2 issues DSNT771I messages, which provide limited information about when the threshold is exceeded.

EXCEPTION_DIAGLEVEL2 values

DB2 issues DSNT772I messages, which provide additional details including the specific profile ID and the specific reason code, when the threshold is exceeded.

WARNING values

DB2 issues messages, depending on the Attribute1 value:

WARNING_DIAGLEVEL1 values

DB2 issues DSNT771I messages, which provide limited information about when the threshold is exceeded.

WARNING_DIAGLEVEL2 values

DB2 issues DSNT772I messages, which provide additional details including the specific profile ID and the specific reason code, when the threshold is exceeded.

Attribute2

An integer value that indicates the threshold of the total number of remote connections that is allowed.

The maximum allowed value is equal to the value of the MAX REMOTE CONNECTED subsystem parameter.

When the specified value is a negative number, this monitor function is disabled and a message is be recorded in the profile attributes history table to indicate that this row is rejected.

Attribute3

NULL

MONITOR IDLE THREADS

Specifies that DB2 monitors the approximate time (in seconds) that an active server thread is allowed to remain idle.

When the specified value is zero, threads are allowed to remain idle indefinitely. When the specified value is a negative number, this monitor function is disabled and a message is be recorded in the profile attributes history table to indicate that the row is rejected.

Attribute1

The type and level of detail for messages issued for monitored threads that meet the conditions specified in the profile. One of the following values:

- EXCEPTION
- EXCEPTION_ DIAGLEVEL1
- EXCEPTION_DIAGLEVEL2
- WARNING
- WARNING_DIAGLEVEL1
- WARNING_ DIAGLEVEL2
- WARNING_MESSAGE_FOR_IDLE_TIMEOUT

The values have the following meanings:

EXCEPTION values

DB2 aborts the thread, pools the DBAT, and terminates the connection if the thread remains idle longer than the specified Attribute2 value.

WARNING values

DB2 issues messages when the thresholds are exceeded.

If the value is WARNING_MESSAGE_FOR_IDLE_TIMEOUT, DB2 issues the following messages:

- DSNT771I displays minimal information to the console.
- DSNT773I displays detailed information about the thread that exceeded the warning threshold. DB2 issues a single DSNT773I message for a thread that remains in an idle state. When a client request message is received, and a COMMIT or ROLLBACK is completed with no resources active past the end of the unit-of-work, DB2 removes the warning against the thread.

WARNING_DIAGLEVEL1 values

DB2 issues DSNT771I messages, which provide limited information about when the threshold is exceeded. This level is also used when the diagnosis level is not specified.

WARNING_DIAGLEVEL2 values

DB2 issues DSNT772I messages, which provide additional details including the specific profile ID and the specific reason code, when the threshold is exceeded.

Attribute2

An integer value that indicates the threshold (in seconds) that a thread is allowed to remain idle. It can be any value that is valid for the IDTHTOIN subsystem parameter.

Threads that meet the criteria of this type of profile are not limited by the value that is specified by the IDTHTOIN subsystem parameter. Consequently you can use MONITOR IDLE THREADS to enable longer idle wait times for certain threads, without increasing the system-wide limit for idle thread timeouts.

A zero value means that matching threads are allowed to remain idle indefinitely. When a negative number is specified, this monitor function is disabled and a message is be recorded in the profile attributes history table to indicate that this row is rejected.

Attribute3
NULL

MONITOR THREADS
Specifies that DB2 monitors the total number of concurrent active threads.

DB2 takes certain actions when the threshold is reached according to the value that is specified in the Attribute1 column. When the total number of threads being queued or suspended reaches the specified threshold, DB2 either issues a message, if a WARNING value is specified, or starts to fail the connection requests with SQLCODE -30041 if an EXCEPTION value is specified.

Attribute1
The type and level of detail for messages issued for monitored threads that meet the conditions specified in the profile. One of the following values:

- EXCEPTION
- EXCEPTION_ DIAGLEVEL1
- EXCEPTION_DIAGLEVEL2
- WARNING
- WARNING_DIAGLEVEL1
- WARNING_ DIAGLEVEL2

The values have the following meanings:

EXCEPTION values
DB2 queues and suspends the threads when the number of active threads exceeds the threshold. When the number of queued threads exceeds the threshold, the action depends on the filtering scope. For certain scopes, DB2 issues a message and begins to fail subsequent connection requests. Additional actions depend on the Attribute1 value:

WARNING values
DB2 issues messages when the total number of active threads exceeds the threshold.

WARNING_DIAGLEVEL1 values
DB2 issues DSNT771I messages, which provide limited information in the message text about the when the threshold is exceeded. This level is also used when the diagnosis level is not specified.

WARNING_DIAGLEVEL2 values
DB2 issues DSNT772I messages, which provide additional details including the specific profile ID and the specific reason code, when the threshold is exceeded.

Attribute2
An integer value that indicates that threshold of the total number of active server threads that is allowed.

The maximum allowed value is equal to the value of the MAX REMOTE ACTIVE subsystem parameter.

When the specified value is a negative number, this monitor function is disabled and a message is be recorded in the profile attributes history table to indicate that this row is rejected.

Attribute3
NULL

NPAGES THRESHOLD
Specifies that DB2 is to set the pages threshold (NPGTHRSH subsystem parameter) to the value that is specified for the Attribute2 column. This value indicates when DB2 should use index access. You can specify one of the following values:

`-1` DB2 is to use index access for as many tables as possible.

`0` DB2 is to select the access path based on the cost. This value is the default.

`1 to` *n*
DB2 should use index access on tables for which the total number of pages (NPAGES) is less than *n*. Ensure that statistics are up to date before you specify a value of 1 or greater.

Attribute1
NULL

Attribute2
An integer 0 or greater

Attribute3
NULL

IO WEIGHTING
Specifies a value of the OPTIOWGT subsystem parameter, which controls how DB2 weights I/O and CPU cost during access path selection. The default value of the OPTIOWGT subsystem parameter is ENABLED. The OPTIOWGT subsystem parameter is deprecated.

Attribute1
One of the following values:

DISABLE
When DISABLE is specified, ENABLE is used instead.

ENABLE
DB2 uses new weights for I/O and CPU that better reflect improved I/O response and caching.

OFF DB2 uses weights for I/O and CPU that reflect I/O and CPU capabilities of older hardware.

Atrribute2
NULL

Attribute3
NULL

SORT_POOL_SIZE
Specifies the sort pool size that a test system uses to model a production system. This attribute value for this keyword corresponds to the value of the SRTPOOL subsystem parameter of the production system. To determine the appropriate value to specify, refer to the value of the PLAN_TABLE.SORTL_POOL_SIZE_EXP column in the EXPLAIN output on the production system.

Attribute1
NULL

Attribute2

A positive integer that is valid for the value SRTPOOL subsystem parameter. When 0 (zero) or a negative number is specified, the profile attribute entry is rejected. When a positive integer that is invalid for the SRTPOOL subsystem parameter is specified, the profile attribute entry is accepted. However, the access paths of SQL statements that are shown in the EXPLAIN tables on the test subsystem might not match the access paths on the production system that is modeled.

Attribute3

NULL

STAR JOIN

Specifies whether DB2 is to use star join processing (the STARJOIN subsystem parameter). You can specify one of the following values for Attribute1:

`DISABLE`

DB2 is not to use star join processing.

`ENABLE`

DB2 is to use star join processing when possible.

Attribute1

DISABLE or ENABLE

Attribute2

NULL

Attribute3

NULL

PSPI

Related tasks:

Monitoring threads and connections by using profiles

Optimizing subsystem parameters for SQL statements by using profiles

Modeling a production environment on a test subsystem

Related reference:

NPGTHRSH in macro DSN6SPRM (DB2 Installation and Migration)

OPTIOWGT in macro DSN6SPRM (DB2 Installation and Migration)

SJTABLES in macro DSN6SPRM (DB2 Installation and Migration)

STAR JOIN QUERIES field (STARJOIN subsystem parameter) (DB2 Installation and Migration)

Related information:

DSNT771I (DB2 Messages)

DSNT772I (DB2 Messages)

DSNT773I (DB2 Messages)

-30041 (DB2 Codes)

SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY

The profile attributes history table, SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY, contains the attributes that were in effect at some point in time.

PSPI

The profile attributes history table, SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY, contains the attributes that were in effect at some point in time. These attributes are defined in the profile attributes table, SYSIBM.DSN_PROFILE_ATTRIBUTES.

However, the ATTRIBUTE_TIMESTAMP column contains the timestamp of the associated PROFILEID in the profile history table. This table is used by optimization tools when diagnosing optimization problems.

The profile tables and related indexes are created when you run job DSNTIJSG during DB2 installation or migration.

The columns for SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY are the same as the columns in SYSIBM.DSN_PROFILE_ATTRIBUTES. Except for the REMARKS column, which is named STATUS instead.

The value of the STATUS column indicates whether the profile was accepted, and when a profile was rejected contains information about the reason for the rejection.

PSPI

Resource limit facility tables

Resource limit facility tables allow you to limit the amount of processor resources, in service units, used by SQL statements in specified contexts.

Resource limits apply only to dynamic SQL statements. The limits that you specify apply to individual SQL statements that qualify for a scope that you define. You can specify reactive or predictive governing, or use a combination of reactive and predictive limits in the same resource limit table.

Important: The following resource limit table formats are either deprecated or not supported in DB2 12 or later releases:

- DSNRLST*xx* table formats earlier than the DB2 Version 8 format are not supported in DB2 12 or later releases.
- DSNRLMT*xx* tables in formats earlier than the DB2 9 format are deprecated in DB2 12.

Related concepts:

Resource limit facility controls

Related tasks:

Limiting resource usage for packages

Setting limits for system resource usage by using the resource limit facility

Controlling resource usage

DSNRLMT*xx* resource limit tables

Resource limit tables can be used to limit the amount of resources used by a certain group of SQL statements. Statements can be limited based on client information, including the application name, user ID, workstation ID, and IP address of the client.

PSPI

Resource limits apply only to the following SQL statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

If both DSNRLMT*xx* and DSNRLST*xx* tables exist, rows in the DSNRLMT*xx* table that match a statement take priority over any matching rows in the DSNRLST*xx* table.

Create table statement

The following statement creates an RLMT resource limit table:

```
CREATE TABLE authid.DSNRLMTxx       (RLFFUNC    CHAR(1)     NOT NULL WITH DEFAULT,
        RLFEUAN    CHAR(32)    NOT NULL WITH DEFAULT,
        RLFEUID    CHAR(16)    NOT NULL WITH DEFAULT,
        RLFEUWN    CHAR(18)    NOT NULL WITH DEFAULT,
        RLFIP      CHAR(254)   NOT NULL WITH DEFAULT,
        ASUTIME    INTEGER,
        RLFASUERR  INTEGER,
        RLFASUWARN INTEGER,
        RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
          IN DSNRLST.DSNRLSxx;
```

In the table name, *xx* is any two-character alphanumeric value. Because the two characters *xx* must be entered as part of the START RLIMIT command, they must be alphanumeric. Special or DBCS characters are not allowed.

Column descriptions

The values in each row define a limit, including the function and scope of each limit. The function of a particular limit is defined by the value of the RLFFUNC column. Other columns specify the scope for the limit defined by the row. For example, you can specify that a limit applies broadly by leaving the RLFEUAN column blank, which means that the row applies to all user IDs, or you can specify limits narrowly by specifying a different row for each user ID for which the function applies. DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on whether reactive or predictive governing is specified. The search order is described under each of those functions.

Table 206. Columns of a DSNRLMTxx resource limit specification table

Column name	Data type	Description
RLFFUNC	CHAR(1) NOT NULL WITH DEFAULT	Specifies how the row is used. The values that have an effect are: **'8'** The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP). **'9'** The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP). All other values are ignored.
RLFEUAN	CHAR(32) NOT NULL WITH DEFAULT	Specifies an application name. A blank value in this column means that the row applies to all application names from the location specified in RLFIP.
RLFEUID	CHAR(16) NOT NULL WITH DEFAULT	Specifies an end user's user ID. A blank value means that the limit specifications in this row apply to every user ID for the location that is specified in RLFIP.
RLFEUWN	CHAR(18) NOT NULL WITH DEFAULT	Specifies an end user's workstation name. A blank value in this column means that the row applies to all workstation names from the location that is specified in RLFIP.
RLFIP	CHAR(254) NOT NULL WITH DEFAULT	The IP address of the location where the request originated. A blank value in this column represents all locations.
ASUTIME	INTEGER	The number of processor service units allowed for any single SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. Use this column for reactive governing. Other possible values and their meanings are: **null** No limit **0 (zero) or a negative value** No SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements are permitted. A relative metric is used so that the resource limit table values do not need to be modified when processors are changed. However, in some cases, DB2 workloads can differ from the measurement averages. In these cases, resource limit table value changes might be necessary.

Table 206. Columns of a DSNRLMTxx resource limit specification table (continued)

Column name	Data type	Description
RLFASUERR	INTEGER	Used for predictive governing (RLFFUNC= '9'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application Other possible values and their effects are: **null** No error threshold **0 (zero) or a negative value** All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE -495.
RLFASUWARN	INTEGER	Used for predictive governing (RLFFUNC= '9'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application. Other possible values and their effects are: **null** No warning threshold **0 (zero) or a negative value** All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE +495. **Important:** Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.
RLF_CATEGORY_B	CHAR(1) NOT NULL WITH DEFAULT	Used for predictive governing (RLFFUNC='9'). Tells the governor the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST_CATEGORY column of the DSN_STATEMNT_TABLE. The acceptable values are: **blank** By default, prepare and execute the SQL statement. **Y** Prepare and execute the SQL statement. **N** Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application. **W** Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.

Search order

DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on which function is being requested (reactive or predictive governing). The search order is described under each of those functions.

Create index statement

To use this table, you must also create an index named *authid*.DSNMRL*xx*, where *xx* represents the same two alphanumeric characters from the table name, in the DSNRLST database.

```
CREATE UNIQUE INDEX authid.DSNMRLxx
       ON authid.DSNRLMTxx
         (RLFFUNC, RLFEUAN DESC, RLFEUID DESC,
          RLFEUWN DESC, RLFIP DESC)
         CLUSTER CLOSE NO;
```

PSPI

Related tasks:

Setting limits for system resource usage by using the resource limit facility

Managing resource limit tables

Limiting resource usage by client information

Related reference:

Job DSNTIJSG (DB2 Installation and Migration)

-START RLIMIT (DB2) (DB2 Commands)

Related information:

+495 (DB2 Codes)

DSNRLST*xx* resource limit tables

Resource limit tables can be used to limit the amount of resource used by certain SQL statements. Statements can be limited based on information about the SQL statement, including the collection ID, package name, authorization ID, and location name of the query.

PSPI

Resource limits apply only to the following statements:

- SELECT
- INSERT
- UPDATE
- MERGE
- TRUNCATE
- DELETE

Resource limits apply only to dynamic SQL statements. Resource limits apply to SQL statement regardless of whether they are issued locally or remotely. The resource limit facility does not control static SQL statements regardless of whether they are issued locally or remotely, and no limits apply to primary or secondary authorization IDs that have installation SYSADM or installation SYSOPR authority.

If both DSNRLMT*xx* and DSNRLST*xx* tables exist, rows in the DSNRLMT*xx* table that match a statement take priority over any matching rows in the DSNRLST*xx* table.

Create table statement

The following statement creates an RLST resource limit table:

```
CREATE TABLE authid.DSNRLSTxx
       (AUTHID   VARCHAR(128)     NOT NULL WITH DEFAULT,
        PLANNAME CHAR(8)       NOT NULL WITH DEFAULT,
        ASUTIME  INTEGER,
        -------3-column format --------
        LUNAME   CHAR(8)       NOT NULL WITH DEFAULT,
        -------4-column format --------
        RLFFUNC  CHAR(1)       NOT NULL WITH DEFAULT,
        RLFBIND  CHAR(1)       NOT NULL WITH DEFAULT,
        RLFCOLLN VARCHAR(128)     NOT NULL WITH DEFAULT,
        RLFPKG   VARCHAR(128)  NOT NULL WITH DEFAULT),
        -------8-column format --------
        RLFASUERR  INTEGER,
        RLFASUWARN INTEGER,
        RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
        -------11-column format --------
         IN DSNRLST.DSNRLSxx;
```

Column descriptions

The values in each row define a limit, including the function and scope of each limit. The function of a particular limit is defined by the value of the RLFFUNC column. Other columns specify the scope for the limit defined by the row. For example, you can specify that a limit broadly by leaving the AUTHID column blank, which means that the row applies to all authorization IDs. Or, you can specify that the limit applies more narrowly by specifying a different row for each authorization ID for which the function applies. Some columns are blank in each row that specifies a valid limit. DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on the type of governing that is specified. The search order is described under each of those functions.

Table 207. Columns of a DSNRLSTxx resource limit specification table

Column name	Data Type	Description
AUTHID	VARCHAR(128) NOT NULL WITH DEFAULT	The resource specification limits apply to this primary authorization ID. A blank means that the limit specifications in this row apply to all authorization IDs for the location that is specified in LUNAME. No limits apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority.
PLANNAME	CHAR(8) NOT NULL WITH DEFAULT	The value of the PLANNAME column must be blank. **Important:** When the DISALLOW_DEFATUL_COLLID subsystem parameter us set to NO, DBRMs that were previously bound into plans are automatically rebound into packages when executed in Version 10. Any row that contains a value for PLANNAME must be modified to specify a blank value for PLANNAME and a value of '2' or '7' for the RLFFUNC column. The modified row must specify the package name (the name of the DBRM) and collection ID (DSN_DEFAULT_COLLID_*plan-name*) for the automatically bound package.

Table 207. Columns of a DSNRLSTxx resource limit specification table (continued)

Column name	Data Type	Description
ASUTIME	INTEGER	The number of processor service units allowed for any single SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. Use this column for reactive governing. Other possible values and their meanings are: **null** No limit **0 (zero) or a negative value** No SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements are permitted. A relative metric is used so that the resource limit table values do not need to be modified when processors are changed. However, in some cases, DB2 workloads can differ from the measurement averages. In these cases, resource limit table value changes might be necessary. For information about how to calculate service units, see Calculating service unit values for resource limit tables.
LUNAME	CHAR(8) NOT NULL WITH DEFAULT	The LU name of the location where the request originated. A blank value in this column represents the local location, *not* all locations. The value PUBLIC represents all local and remote DBMS locations in the network; these locations do not need to be DB2 subsystems. PUBLIC is the only value for TCP/IP connections.
RLFFUNC	CHAR(1) NOT NULL WITH DEFAULT	**'1'** The row reactively governs bind operations. **'2'** The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name. **'3'** The row disables query I/O parallelism. Query I/O parallelism is deprecated and is likely to be removed in a future release. **'4'** The row disables query CP parallelism. **'5'** The row disables Sysplex query parallelism. Sysplex query parallelism is deprecated and is likely to be removed in a future release. **'7'** The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name. All other values are ignored.
RLFBIND	CHAR(1) NOT NULL WITH DEFAULT	Shows whether bind operations are allowed. An 'N' implies that bind operations are not allowed. Any other value means that bind operations are allowed. This column is used only if RLFFUNC is set to '1'.
RLFCOLLN	VARCHAR(128) NOT NULL WITH DEFAULT	Specifies a package collection. A blank value in this column means that the row applies to all package collections from the location that is specified in LUNAME. Qualify by collection name only if the statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC='1', RLFCOLLN must be blank.
RLFPKG	VARCHAR(128) NOT NULL WITH DEFAULT	Specifies a package name. A blank value in this column means that the row applies to all packages from the location that is specified in LUNAME. If RLFFUNC='1', RLFPKG must be blank.

Table 207. Columns of a DSNRLSTxx resource limit specification table (continued)

Column name	Data Type	Description
RLFASUERR	INTEGER	Used for predictive governing (RLFFUNC= '7'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application. Other possible values and their effects are: **null** No error threshold **0 (zero) or a negative value** All dynamicSELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE -495.
RLFASUWARN	INTEGER	Used for predictive governing (RLFFUNC= '7'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application. Other possible values and their effects are: **null** No warning threshold **0 (zero) or a negative value** All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE +495. **Important:** Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.
RLF_CATEGORY_B	CHAR (1) NOT NULL WITH DEFAULT	Used for predictive governing (RLFFUNC= '7'). Tells the resource limit facility the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST_CATEGORY column of the DSN_STATEMNT_TABLE. The acceptable values are: **blank** By default, prepare and execute the SQL statement. **Y** Prepare and execute the SQL statement. **N** Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application. **W** Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.

Create index statement

The following statement creates an index that is required for use with RLST resource limit tables:

```
CREATE UNIQUE INDEX authid.DSNARLxx
       ON authid.DSNRLSTxx
         (RLFFUNC, AUTHID DESC, PLANNAME DESC,
          RLFCOLLN DESC, RLFPKG DESC, LUNAME DESC)
         CLUSTER CLOSE NO;
```

Note: The value of *xx* in the index name must match the *xx* in the table name (DSNRLST*xx*), and the index must be a descending index.

PSPI

Related tasks:

Setting limits for system resource usage by using the resource limit facility

Managing resource limit tables

Limiting resource usage for packages

Related reference:

-START RLIMIT (DB2) (DB2 Commands)

Related information:

-495 (DB2 Codes)

+495 (DB2 Codes)

Information resources for DB2 10 for z/OS and related products

Information about DB2 10 for z/OS and products that you might use in conjunction with DB2 10 is available online in IBM Knowledge Center or on library websites.

Obtaining DB2 for z/OS publications

Current DB2 10 for z/OS publications are available from the following websites:

http://www-01.ibm.com/support/docview.wss?uid=swg27019288

Links to IBM Knowledge Center and the PDF version of each publication are provided.

DB2 for z/OS publications are also available for download from the IBM Publications Center (http://www.ibm.com/shop/publications/order).

In addition, books for DB2 for z/OS are available on a CD-ROM that is included with your product shipment:

- DB2 10 for z/OS Licensed Library Collection, LK5T-7390, in English. The CD-ROM contains the collection of books for DB2 10 for z/OS in PDF format. Periodically, IBM refreshes the books on subsequent editions of this CD-ROM.

Installable information center

You can download or order an installable version of the Information Management Software for z/OS Solutions Information Center, which includes information about DB2 10 for z/OS, QMF, IMS, and many DB2 Tools for z/OS products. You can install this information center on a local system or on an intranet server. For more information, see http://www-01.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc/src/alltoc/installabledzic.html.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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

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

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

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

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM websites 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 which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

This information contains 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 actual people or business enterprises 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 shown below:

© (*your company name*) (*year*).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. (*enter the year or years*).

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This information is intended to help you to plan for and administer DB2 10 for z/OS. This information also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by DB2 10 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 10 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

GUPI

General-use Programming Interface and Associated Guidance Information...

GUPI

Product-sensitive Programming Interface and Associated Guidance Information

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

PSPI

Product-sensitive Programming Interface and Associated Guidance Information...

PSPI

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered marks 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: http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Glossary

The glossary is available in IBM Knowledge Center.

See the Glossary topic for definitions of DB2 for z/OS terms.

Index

Product Number: 5605-DB2
5697-P31

Printed in USA

SC19-2978-17

Spine information:

DB2 10 for z/OS

Managing Performance

IBM