IBM i
Version 7.2

Database
Performance and Query Optimization

IBM

Note

Before using this information and the product it supports, read the information in "Notices" on page 647.

Contents

Database performance and query optimization

The goal of database performance tuning is to minimize the response time of your queries by making the best use of your system resources. The best use of these resources involves minimizing network traffic, disk I/O, and CPU time. This goal can only be achieved by understanding the logical and physical structure of your data, the applications used on your system, and how the conflicting uses of your database might affect performance.

The best way to avoid performance problems is to ensure that performance issues are part of your ongoing development activities. Many of the most significant performance improvements are realized through careful design at the beginning of the database development cycle. To most effectively optimize performance, you must identify the areas that yield the largest performance increases over the widest variety of situations. Focus your analysis on these areas.

Many of the examples within this publication illustrate a query written through either an SQL or an OPNQRYF query interface. The interface chosen for a particular example does not indicate an operation exclusive to that query interface, unless explicitly noted. It is only an illustration of one possible query interface. Most examples can be easily rewritten into whatever query interface that you prefer.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 645.

What's new for IBM i 7.2

The following information was added or updated in this release of the information:

- Support for Row and column access controls:
  - “Row and column access control (RCAC)” on page 101
  - QAQQINI options for RCAC: “QAQQINI query options” on page 176
- DB2 for i services outlines many system-provided views, procedures, and functions. “DB2 for i Services” on page 264
- IBM i Services outlines many system services that can be accessed through system-provided SQL views, procedures, and functions. “IBM i Services” on page 301
- Improved query identification matching advised indexes with existing Plan Cache queries.
  - “Index advisor” on page 145
  - “Displaying index advisor information” on page 147
- Allow for host variable selectivity checks at pseudo open time:
  - “Reducing the number of open operations” on page 247
  - “QAQQINI query options” on page 176
- “Controlling queries dynamically with the query options file QAQQINI” on page 172
- Temporary indexes as a source of optimizer statistics: “Temporary index” on page 44
- Improved query optimization I/O cost estimates based on IPL determined disk I/O analysis: “Single table optimization” on page 56
- Support for Solid State Drives: “Solid State Drives” on page 57
- “Index advisor column descriptions” on page 150
- “Database manager indexes advised system table” on page 148
- New system limit for index size 1.7 Terabytes: “QSYS2.Health_Size_Limits ()” on page 121
- Index advice generation now handles OR predicates: “Index advice and OR predicates” on page 145
- SQE Plan Cache default auto sizing vs explicit size designation:

  - "Plan cache" on page 9
  - "Accessing the SQL plan cache with SQL stored procedures" on page 163
- EVI INCLUDE supports grouping set queries "Recommendations for EVI use" on page 217

What's new

The following revisions or additions have been made to the Performance and query optimization documentation since the first 7.2 publication:

- **April 2019 update**
  - The maximum table size has been added as a tracked system limit and as a limit that sends alerts: "System Health Services" on page 429, "System limit alerts" on page 431
- **August 2018 update**
  - **New services**
    - GENERATE_SQL_OBJECTS procedure: "GENERATE_SQL_OBJECTS procedure" on page 292
    - JOB_DESCRIPTION_INFO view: "JOB_DESCRIPTION_INFO view" on page 449
    - OUTPUT_QUEUE_ENTRIES_BASIC view: "OUTPUT_QUEUE_ENTRIES_BASIC view" on page 409
  - **Updated services**
    - ACTIVE_JOB_INFO table function optionally returns more detailed information: "ACTIVE_JOB_INFO table function" on page 436
    - NETSTAT_INFO view and NETSTATE_JOB_INFO view return port names from service table entries: "NETSTAT_INFO view" on page 311 and "NETSTAT_JOB_INFO view" on page 326
    - PARSE_STATEMENT table function supports some DDL references: "PARSE_STATEMENT table function" on page 266
- **October 2017 update**
  - **New services**
    - ASP_INFO view: "ASP_INFO view" on page 417
    - ASP_VARY_INFO view: "ASP_VARY_INFO view" on page 423
    - JOB_QUEUE_INFO view: "JOB_QUEUE_INFO view" on page 470
    - STACK_INFO table function: "STACK_INFO table function" on page 306
  - **Updated services**
    - DISPLAY_JOURNAL and HISTORY_LOG_INFO include syslog information: "DISPLAY_JOURNAL table function" on page 342 and "HISTORY_LOG_INFO table function" on page 363
    - OVERRIDE_QAQQINI procedure has been fully documented: "OVERRIDE_QAQQINI procedure" on page 265
    - System limit notifications: "System limit alerts" on page 431
- **March 2017 update**
  - **New services**
    - AUTHORIZATION_LIST_INFO view: "AUTHORIZATION_LIST_INFO view" on page 384
    - AUTHORIZATION_LIST_USER_INFO view: "AUTHORIZATION_LIST_USER_INFO view" on page 386
    - OBJECT_PRIVILEGES view: "OBJECT_PRIVILEGES view" on page 390
    - MESSAGE_QUEUE_INFO view: "MESSAGE_QUEUE_INFO view" on page 369
    - LICENSE_EXPIRATION_CHECK procedure: "LICENSE_EXPIRATION_CHECK procedure" on page 372
    - SET_PASE_SHELL_INFO procedure: "SET_PASE_SHELL_INFO procedure" on page 305
  - **Updated services**

- USER_INFO has new columns for supplemental group profile information and the PASE shell: “USER_INFO view” on page 394
- LICENSE_INFO view has a new column indicating the install status: “LICENSE_INFO view” on page 372
- RESET_TABLE_INDEX_STATISTICS procedure has a new option to remove rows from the index advice tracking table: “RESET_TABLE_INDEX_STATISTICS procedure” on page 276

- **November 2016 update**
  - STATEMENT DETERMINISTIC option has been added for functions: “QAQQINI query options” on page 176
  - **New services**
    - HISTORY_LOG_INFO table function: “HISTORY_LOG_INFO table function” on page 363
    - JOB_INFO table function: “JOB_INFO table function” on page 457
    - PARSE_STATEMENT table function: “PARSE_STATEMENT table function” on page 266
  - **Updated services**
    - DISPLAY_JOURNAL table function honors row and column access control: “DISPLAY_JOURNAL table function” on page 342
    - GET_JOB_INFO table function has new columns for prestart job information: “GET_JOB_INFO table function” on page 456
    - GROUP_PTF_CURRENCY view returns a new value to indicate PTFs will be current with the next IPL: “GROUP_PTF_CURRENCY view” on page 375
    - GROUP_PTF_CURRENCY and GROUP_PTF_DETAILS views have been updated to access a new XML feed: “GROUP_PTF_CURRENCY view” on page 375 and “GROUP_PTF_DETAILS view” on page 376
    - OBJECT_STATISTICS table function added an option to efficiently return a list of libraries: “OBJECT_STATISTICS table function” on page 360
- **April 2016 update**
  - **New services**
    - ENVIRONMENT_VARIABLE_INFO view: “ENVIRONMENT_VARIABLE_INFO view” on page 302
    - OUTPUT_QUEUE_INFO view: “OUTPUT_QUEUE_INFO view” on page 411
    - SERVICES_INFO table and DB2 PTF Group level dependency information: “SERVICES_INFO table” on page 303
  - **Updated services**
    - DISPLAY_JOURNAL table function accepts ending values as input parameters to limit the entries returned: “DISPLAY_JOURNAL table function” on page 342
    - NETSTAT_INFO view has been updated to return more information: “NETSTAT_INFO view” on page 311
    - NETSTAT_INTERFACE_INFO view has been updated to return more information: “NETSTAT_INTERFACE_INFO view” on page 318
    - NETSTAT_JOB_INFO view has been updated to return more information: “NETSTAT_JOB_INFO view” on page 326
    - NETSTAT_ROUTE_INFO view has been updated to return more information: “NETSTAT_ROUTE_INFO view” on page 327
    - SERVER_SBS_ROUTING view shows information about more servers: “SERVER_SBS_ROUTING view” on page 337
    - SET_SERVER_SBS_ROUTING procedure allows you to configuring more servers: “SET_SERVER_SBS_ROUTING procedure” on page 334
    - SYSLIMITS view returns more information about each object: “SYSLIMITS view” on page 433

- An additional limit is tracked: Maximum extended dynamic package size: "System Health Services" on page 429

- **October 2015 update**
  - **New services**
    - GROUP_PTF_DETAILS view: "GROUP_PTF_DETAILS view" on page 376
    - LICENSE_INFO view: "LICENSE_INFO view" on page 372
    - MEDIA_LIBRARY_INFO view: "MEDIA_LIBRARY_INFO view" on page 425
    - MEMORY_POOL table function: "MEMORY_POOL table function" on page 474
    - MEMORY_POOL_INFO view: "MEMORY_POOL_INFO view" on page 476
    - NETSTAT_INFO view: "NETSTAT_INFO view" on page 311
    - NETSTAT_INTERFACE_INFO view: "NETSTAT_INTERFACE_INFO view" on page 318
    - NETSTAT_JOB_INFO view: "NETSTAT_JOB_INFO view" on page 326
    - NETSTAT_ROUTE_INFO view: "NETSTAT_ROUTE_INFO view" on page 327
    - OBJECT_LOCK_INFO view: "OBJECT_LOCK_INFO view" on page 478
    - OUTPUT_QUEUE_ENTRIES table function: "OUTPUT_QUEUE_ENTRIES table function" on page 399
    - OUTPUT_QUEUE_ENTRIES view: "OUTPUT_QUEUE_ENTRIES view" on page 404
    - RECORD_LOCK_INFO view: "RECORD_LOCK_INFO view" on page 480
    - SYSTEM_STATUS table function: "SYSTEM_STATUS table function" on page 485
    - SYSTEM_STATUS_INFO view: "SYSTEM_STATUS_INFO view" on page 487
  - **Updated services**
    - ACTIVE_JOB_INFO table function has been updated to return elapsed time: "ACTIVE_JOB_INFO table function" on page 436
    - DATABASE_MONITOR_INFO view has been updated to describe new filter values: "DATABASE_MONITOR_INFO view" on page 271
    - ENV_SYS_INFO view has been updated to return the total configured memory: "ENV_SYS_INFO view" on page 311
    - GET_JOB_INFO table function has been updated to return the client IP address: "GET_JOB_INFO table function" on page 456
    - SET_SERVER_SBS_ROUTING procedure allows you to configuring the remote command server: "SET_SERVER_SBS_ROUTING procedure" on page 334
- **May 2015 update**
  - Additional information was added to QQI1 - Insert unique count in the database monitor 1000 record. For details, see: "Database monitor view 1000 - SQL Information" on page 498
  - Additional options were added to the QAQQINI query option Memory_Pool_Preference. For details, see: "QAQQINI query options" on page 176
  - CLEAR_PLAN_CACHE procedure. For details, see: "CLEAR_PLAN_CACHE" on page 169
  - **New services**
    - ACTIVE_JOB_INFO table function: "ACTIVE_JOB_INFO table function" on page 436
    - DATABASE_MONITOR_INFO view: "DATABASE_MONITOR_INFO view" on page 271
    - DRDA_AUTHENTICATION_ENTRY_INFO view: "DRDA_AUTHENTICATION_ENTRY_INFO view" on page 388
    - JVM_INFO view: "JVM_INFO view" on page 339
    - SCHEDULED_JOB_INFO view: "SCHEDULED_JOB_INFO view" on page 481
    - SERVER_SBS_ROUTING view: "SERVER_SBS_ROUTING view" on page 337
    - SET_JVM procedure: "SET_JVM procedure" on page 341

- SET_SERVER_SBS_ROUTING procedure: “SET_SERVER_SBS_ROUTING procedure” on page 334
   - **Updated services**
     - GET_JOB_INFO table function has been updated to return additional SQL information for a job: “GET_JOB_INFO table function” on page 456
     - OBJECT_STATISTICS table function has a new optional parameter to specify the name of the object to return. It will also return the long SQL name for an object and has new columns to return the text, the long schema name, and the SQL type of an object: “OBJECT_STATISTICS table function” on page 360
     - System Health Services has been updated to track index limits: “System Health Services” on page 429
- **October 2014 update**
   - **Updates to the QAQQINI query options topic**

     For details, see “QAQQINI query options” on page 176.
   - **Memory preference controls enhanced for SQL**

     For details, see “Memory preference controls” on page 58
   - The database monitor topic has been updated: “Monitoring your queries using the Database Monitor ” on page 129
   - The SQL Plan Cache topic has been updated: “Optimizing performance using the Plan Cache” on page 156
   - **New services**
     - LIBRARY_LIST_INFO view: “LIBRARY_LIST_INFO view” on page 360
     - REPLY_LIST_INFO view: “REPLY_LIST_INFO view” on page 371
     - JOURNAL_INFO view: “JOURNAL_INFO view” on page 351
     - GROUP_PTF_CURRENCY view: “GROUP_PTF_CURRENCY view” on page 375
     - JOBLOG_INFO table function: “JOBLOG_INFO table function” on page 367
   - **Tracking of additional file system limits**

     For details, see “System Health Services” on page 429

How to see what's new or changed

To help you see where technical changes have been made, this information uses:

- The » image to mark where new or changed information begins.
- The « image to mark where new or changed information ends.

To find other information about what's new or changed this release, see the Memo to users.

PDF file for Database performance and query optimization

View and print a PDF of this information.

To view or download the PDF version of this document, select Database performance and query optimization.

Other information

You can also view or print any of the following PDF files:

- Preparing for and Tuning the SQL Query Engine on DB2® for i5/OS
- SQL Performance Diagnosis on IBM® DB2 Universal Database for iSeries

.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the preceding link).
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDF files. You can download a free copy from Adobe (http://get.adobe.com/reader/).

Query engine overview

IBM DB2 for i provides two query engines to process queries: Classic Query Engine (CQE) and SQL Query Engine (SQE).

The CQE processes queries originating from non-SQL interfaces: OPNQRYF, Query/400, and QQQQry API. SQL-based interfaces, such as ODBC, JDBC, CLI, Query Manager, Net.Data®, RUNSQLSTM, and embedded or interactive SQL, run through the SQE. For ease of use, the routing decision for processing the query by either CQE or SQE is pervasive and under the control of the system. The requesting user or application program cannot control or influence this behavior. However, a better understanding of the engines and process that determines which path a query takes can give you a better understanding of query performance.

Within SQE, several more components were created and other existing components were updated. Additionally, new data access methods are possible with SQE that are not supported under CQE.

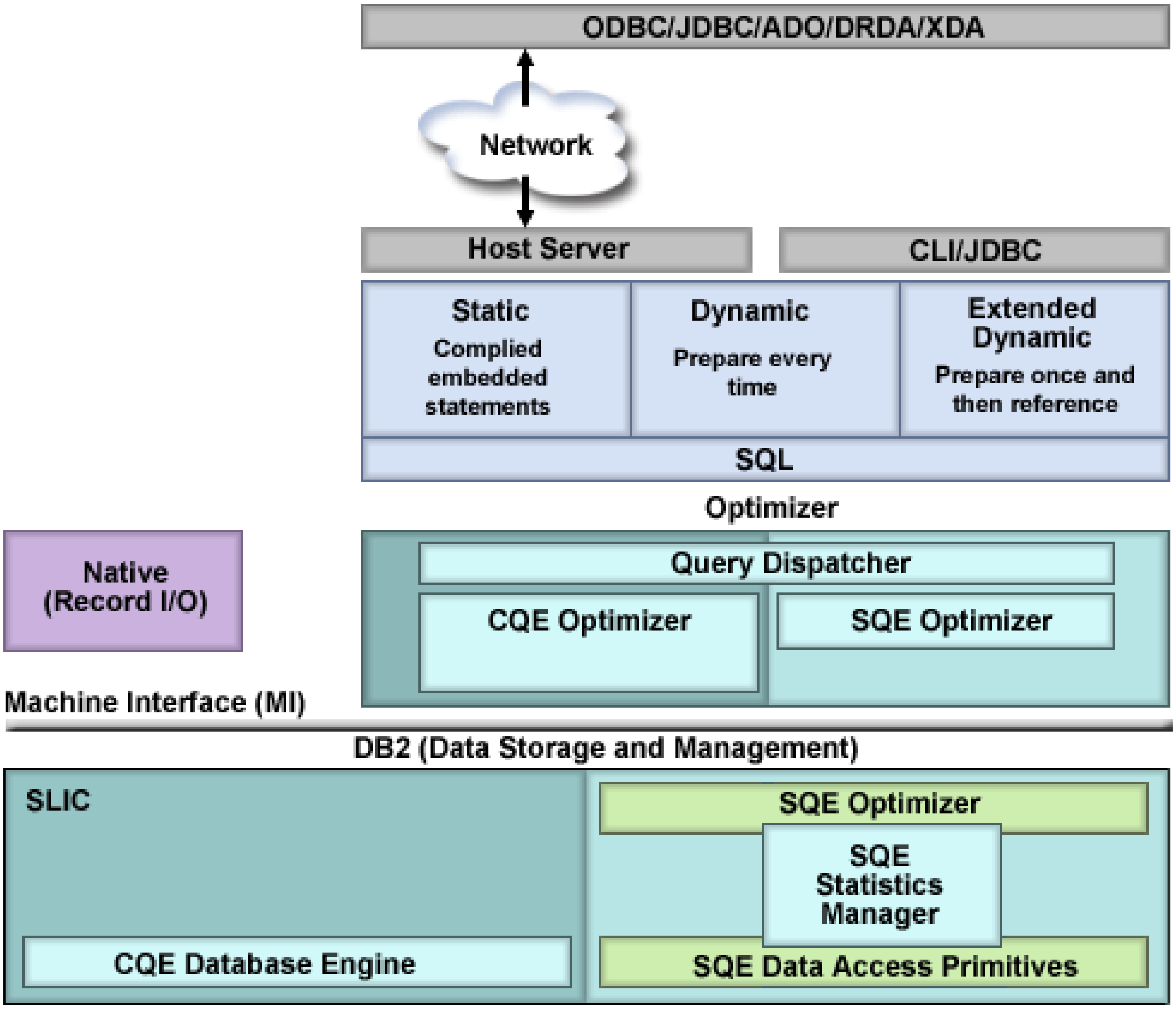
Related information

Embedded SQL programming
SQL programming
Query (QQQQRY) API
Open Query File (OPNQRYF) command
Run SQL Statements (RUNSQLSTM) command

SQE and CQE engines

It is important to understand the implementation differences of query management and processing in CQE versus SQE.

The following figure shows an overview of the IBM DB2 for i architecture. It shows the delineation between CQE and SQE, how query processing is directed by the query dispatcher, and where each SQE component fits. The functional separation of each SQE component is clearly evident. This division of responsibility enables IBM to more easily deliver functional enhancements to the individual components of SQE, as and when required. Notice that most of the SQE Optimizer components are implemented below the MI. This implementation translates into enhanced performance efficiency.



As seen in the previous graphic, the query runs from any query interface to the optimizer and the query dispatcher. The query dispatcher determines whether the query is implemented with CQE or SQE.

Query dispatcher

The function of the dispatcher is to route the query request to either CQE or SQE, depending on the attributes of the query. All queries are processed by the dispatcher. It cannot be bypassed.

Currently, the dispatcher routes queries to SQE unless it finds that the query references or contains any of the following:

- INSERT WITH VALUES statement or the target of an INSERT with subselect statement
- Tables with Read triggers
- Read-only queries with more than 1000 dataspaces, or updatable queries with more than 256 dataspaces.
- DB2 Multisystem tables
- QQQQry API

For other non-SQL queries, for example Query/400 or OPNQRYF, the routing of the query can be controlled by the QAQQINI SQE_NATIVE_ACCESS option. See "table 46".

Related reference

MQT supported function

Although an MQT can contain almost any query, the optimizer only supports a limited set of query functions when matching MQTs to user specified queries. The user-specified query and the MQT query must both be supported by the SQE optimizer.

Statistics manager

In CQE, the retrieval of statistics is a function of the Optimizer. When the Optimizer needs to know information about a table, it looks at the table description to retrieve the row count and table size. If an index is available, the Optimizer might extract information about the data in the table. In SQE, the collection and management of statistics is handled by a separate component called the statistics manager. The statistics manager leverages all the same statistical sources as CQE, but adds more sources and capabilities.

The statistics manager does not actually run or optimize the query. Instead, it controls the access to the metadata and other information that is required to optimize the query. It uses this information to answer questions posed by the query optimizer. The statistics manager always provides answers to the optimizer. In cases where it cannot provide an answer based on actual existing statistics information, it is designed to provide a predefined answer.

The Statistics manager typically gathers and tracks the following information:

Cardinality of values
: The number of unique or distinct occurrences of a specific value in a single column or multiple columns of a table.

Selectivity
: Also known as a histogram, this information is an indication of how many rows are selected by any given selection predicate or combination of predicates. Using sampling techniques, it describes the selectivity and distribution of values in a given column of the table.

Frequent values
: The top *nn* most frequent values of a column together with a count of how frequently each value occurs. This information is obtained by using statistical sampling techniques. Built-in algorithms eliminate the possibility of data skewing. For example, NULL values and default values that can influence the statistical values are not taken into account.

Metadata information
: Includes the total number of rows in the table, indexes that exist over the table, and which indexes are useful for implementing the particular query.

Estimate of IO operation
: An estimate of the amount of IO operations that are required to process the table or the identified index.

The Statistics manager uses a hybrid approach to manage database statistics. Most of this information can be obtained from existing indexes. In cases where the required statistics cannot be gathered from existing indexes, statistical information is constructed on single columns of a table and stored internally. By default, this information is collected automatically by the system, but you can manually control the collection of statistics. Unlike indexes, however, statistics are not maintained immediately as data in the tables change.

Related reference

Collecting statistics with the statistics manager
The collection of statistics is handled by a separate component called the statistics manager. Statistical information can be used by the query optimizer to determine the best access plan for a query. Since the

query optimizer bases its choice of access plan on the statistical information found in the table, it is important that this information is current.

Global Statistics Cache

In SQE, the DB2 Statistics Manager stores actual row counts into a Global Statistics Cache. In this manner, the Statistics Manager refines its estimates over time as it learns where estimates have deviated from actual row counts.

Both completed queries and currently executing queries might be inspected by the "Adaptive Query Processing" on page 97 (AQP) task, which compares estimated row counts to actual row counts. If there are any significant discrepancies, the AQP task notifies the DB2 Statistics Manager (SM). The SM stores this actual row count (also called observed row count) into a Global Statistics Cache (GSC).

If the query which generated the observed statistic in the GSC is reoptimized, the actual row count estimate is used in determining a new query plan. Further, if a different query asks for the same or a similar row count, the SM could return the stored actual row count from the GSC. Faster query plans can be generated by the query optimizer.

Typically, observed statistics are for complex predicates such as with a join. A simple example is a query joining three files A, B, and C. There is a discrepancy between the estimate and actual row count of the join of A and B. The SM stores an observed statistic into the GSC. Later, if a different join query of A, B, and Z is submitted, the SM recalls the observed statistic of the A and B join. The SM considers that observed statistic in its estimate of the A, B, and Z join.

The Global Statistics Cache is an internal DB2 object, and the contents of it are not directly observable.

Plan cache

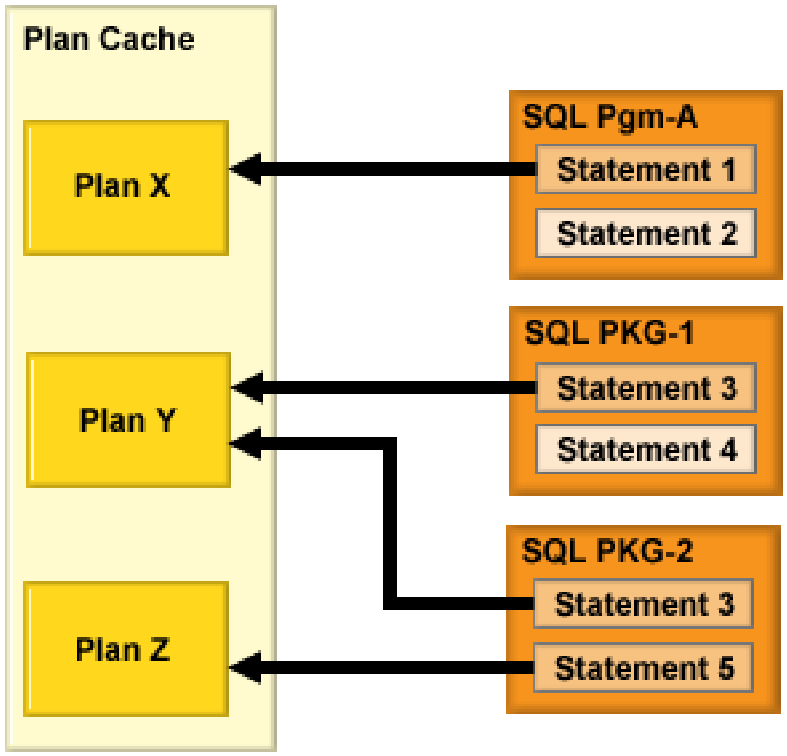
The plan cache is a repository that contains the access plans for queries that were optimized by SQE.

Access plans generated by CQE are not stored in the plan cache; instead, they are stored in SQL packages, the system-wide statement cache, and job cache. The purposes of the plan cache are to:

- Facilitate the reuse of a query access plan when the same query is re-executed
- Store runtime information for subsequent use in future query optimizations
- Provide performance information for analysis and tuning

Once an access plan is created, it is available for use by all users and all queries, regardless of where the query originates. Furthermore, when an access plan is tuned, for example, when creating an index, all queries can benefit from this updated access plan. This updated access plan eliminates the need to reoptimize the query, resulting in greater efficiency.

The following graphic shows the concept of reusability of the query access plans stored in the plan cache:



As shown in the previous graphic, statements from packages and programs are stored in unique plans in the plan cache. If Statement 3 exists in both SQL package 1 and SQL package 2, the plan is stored once in the plan cache. The plan cache is interrogated each time a query is executed. If an access plan exists that satisfies the requirements of the query, it is used to implement the query. Otherwise a new access plan is created and stored in the plan cache for future use.

The plan cache is automatically updated with new query access plans as they are created. When new statistics or indexes become available, an existing plan is updated the next time the query is run. The plan cache is also automatically updated by the database with runtime information as the queries are run.

Each plan cache entry contains the original query, the optimized query access plan, and cumulative runtime information gathered during the runs of the query. In addition, several instances of query runtime objects are stored with a plan cache entry. These runtime objects are the real executable objects and temporary storage containers (hash tables, sorts, temporary indexes, and so on) used to run the query.

By default the SQE Plan Cache will auto adjust from an initial threshold size of 512 MB to an internally managed maximum. Automatic management of the SQL Plan Cache Threshold Size by the system will not take effect if the plan cache threshold size is explicitly set on the system. See the SQL plan cache properties topic for more information: rzajqplancacheprops.dita

- When processing is initiated to remove plans in the cache due to size constraint, the efficiency rating of the cache is checked. If the rating is too low, the database will automatically increase the plan cache size.
- The plan cache auto-sizing maximum size will not exceed a small percentage of free storage on the system.
- The plan cache auto-sizing will decrease the size if the temporary storage on the machine exceeds a certain percentage.
- The auto-sized adjusted threshold value does not survive an IPL. The default plan cache size is used after an IPL and auto sizing begins again.
- To reset an explicitly set plan cache size in order to allow auto-sizing to take effect, set the plan cache size to zero.

Example:

```
CALL qsys2.change_plan_cache_size(0)
```

When the plan cache exceeds its designated size, a background task is automatically scheduled to remove plans from the plan cache. Access plans are deleted based upon age, how frequently it is used, and how much cumulative resources (CPU/IO) were consumed.

The total number of access plans stored in the plan cache depends largely upon the complexity of the SQL statements that are being executed. The plan cache is cleared when a system Initial Program Load (IPL) is performed.

Multiple access plans for a single SQL statement can be maintained in the plan cache. Although the SQL statement is the primary key into the plan cache, different environmental settings can cause additional access plans to be stored. Examples of these environmental settings include:

- Different SMP Degree settings for the same query
- Different library lists specified for the query tables
- Different settings for the share of available memory for the job in the current pool
- Different ALWCPYDTA settings
- Different selectivity based on changing host variable values used in selection (WHERE clause)

Currently, the plan cache can maintain a maximum of three different access plans for the same SQL statement. As new access plans are created for the same SQL statement, older access plans are discarded to make room for the new access plans. There are, however, certain conditions that can cause an existing access plan to be invalidated. Examples of these conditions include:

- Specifying REOPTIMIZE_ACCESS_PLAN(*YES) or (*FORCE) in the QAQQINI table or in Run SQL Scripts
- Deleting or recreating the table that the access plan refers to
- Deleting an index that is used by the access plan

Related reference

Effects of the ALWCPYDTA parameter on database performance
Some complex queries can perform better by using a sort or hashing method to evaluate the query instead of using or creating an index.

Changing the attributes of your queries
You can modify different types of query attributes for a job with the **Change Query Attributes (CHGQRYA)** CL command. You can also use the System i Navigator Change Query Attributes interface.

Optimizing performance using the Plan Cache
The SQL Plan Cache contains a wealth of information about the SQE queries being run through the database. Its contents are viewable through the System i Navigator GUI interface. Certain portions of the plan cache can also be modified.

Data access methods

Data access methods are used to process queries and access data.

In general, the query engine has two kinds of raw material with which to satisfy a query request:

- The database objects that contain the data to be queried
- The executable instructions or operations to retrieve and transform the data into usable information

There are only two types of permanent database objects that can be used as source material for a query — tables and indexes. Indexes include binary radix and encoded vector indexes.

In addition, the query engine might need to create temporary objects to hold interim results or references during the execution of an access plan. The DB2 Symmetric Multiprocessing feature provides the optimizer with additional methods for retrieving data that include parallel processing. Finally, the optimizer uses certain methods to manipulate these objects.

Permanent objects and access methods

There are three basic types of access methods used to manipulate the permanent and temporary database objects -- Create, Scan, and Probe.

The following table lists each object and the access methods that can be performed against that object. The symbols shown in the table are the icons used by Visual Explain.

Table 1. Permanent object data access methods

Permanent objects	Scan operations	Probe operations
Table	Table scan	Table probe
Radix index	Radix index scan	Radix index probe
Encoded vector index	Encoded vector index symbol table scan	Encoded vector index probe

Table

An SQL table or physical file is the base object for a query. It represents the source of the data used to produce the result set for the query. It is created by the user and specified in the FROM clause (or OPNQRYF FILE parameter).

The optimizer determines the most efficient way to extract the data from the table in order to satisfy the query. These ways could include scanning or probing the table or using an index to extract the data.

Visual explain icon:

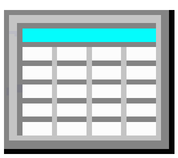


Table scan

A table scan is the easiest and simplest operation that can be performed against a table. It sequentially processes all the rows in the table to determine if they satisfy the selection criteria specified in the query. It does this processing in a way to maximize the I/O throughput for the table.

A table scan operation requests large I/Os to bring as many rows as possible into main memory for processing. It also asynchronously pre-fetches the data to make sure that the table scan operation is never waiting for rows to be paged into memory. Table scan however, has a disadvantage in it has to process all the rows in order to satisfy the query. The scan operation itself is efficient if it does not need to perform the I/O synchronously.

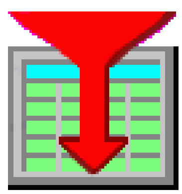
Table 2. Table scan attributes

Data access method	Table scan
Description	Reads all the rows from the table and applies the selection criteria to each of the rows within the table. The rows in the table are processed in no guaranteed order, but typically they are processed sequentially.
Advantages	• Minimizes page I/O operations through asynchronous pre-fetching of the rows since the pages are scanned sequentially • Requests a larger I/O to fetch the data efficiently

Table 2. Table scan attributes (continued)	
Data access method	**Table scan**
Considerations	• All rows in the table are examined regardless of the selectivity of the query • Rows marked as deleted are still paged into memory even though none are selected. You can reorganize the table to remove deleted rows.
Likely to be used	• When expecting many rows returned from the table • When the number of large I/Os needed to scan is fewer than the number of small I/Os required to probe the table

Example SQL statement	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01'AND 'E01'`
	`OPTIMIZE FOR ALL ROWS`

Messages indicating use	• Optimizer Debug:
	`CPI4329 — Arrival sequence was used for file EMPLOYEE`
	• PRTSQLINF:
	`SQL4010 — Table scan access for table 1.`

SMP parallel enabled	Yes
Also referred to as	Table Scan, Preload
Visual Explain icon	

Related concepts

Nested loop join implementation

Db2 for i provides a **nested loop** join method. For this method, the processing of the tables in the join are ordered. This order is called the **join order**. The first table in the final join order is called the **primary table**. The other tables are called **secondary tables**. Each join table position is called a **dial**.

Table probe

A table probe operation is used to retrieve a specific row from a table based upon its row number. The row number is provided to the table probe access method by some other operation that generates a row number for the table.

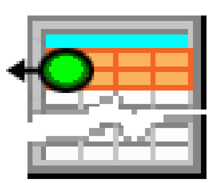
This can include index operations as well as temporary row number lists or bitmaps. The processing for a table probe is typically random. It requests a small I/O to retrieve only the row in question and does not attempt to bring in any extraneous rows. This method leads to efficient processing for smaller result sets because only rows needed to satisfy the query are processed, rather than scanning all rows.

However, since the sequence of the row numbers is not known in advance, little pre-fetching can be performed to bring the data into main memory. This randomness can result in most of the I/Os associated with table probe to be performed synchronously.

Table 3. Table probe attributes	
Data access method	**Table probe**
Description	Reads a single row from the table based upon a specific row number. A random I/O is performed against the table to extract the row.
Advantages	• Requests smaller I/Os to prevent paging rows into memory that are not needed • Can be used with any access method that generates a row number for the table probe to process
Considerations	Because of the synchronous random I/O the probe can perform poorly when many rows are selected
Likely to be used	• When row numbers (from indexes or temporary row number lists) are used, but data from the underlying table is required for further processing of the query • When processing any remaining selection or projection of the values

Example SQL statement	`CREATE INDEX X1 ON Employee (LastName)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`AND LastName IN ('Smith', 'Jones', 'Peterson')`
	`OPTIMIZE FOR ALL ROWS`

Messages indicating use	There is no specific message that indicates the use of a table probe. These example messages illustrate the use of a data access method that generates a row number used to perform the table probe.
	• Optimizer Debug:
	`CPI4328 — Access path of file X1 was used by query`
	• PRTSQLINF:
	`SQL4008 — Index X1 used for table 1.`
	
	`SQL4011 — Index scan-key row positioning (probe) used on table 1.`

SMP parallel enabled	Yes
Also referred to as	Table Probe, Preload
Visual Explain icon	

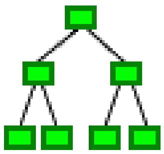
Radix index

An SQL index (or keyed sequence access path) is a permanent object that is created over a table. The index is used by the optimizer to provide a sequenced view of the data for a scan or probe operation.

The rows in the tables are sequenced in the index based upon the key columns specified on the creation of the index. When the optimizer matches a query to index key columns, it can use the index to help satisfy query selection, ordering, grouping, or join requirements.

Typically, using an index also includes a table probe to provide access to columns needed to satisfy the query that cannot be found as index keys. If all the columns necessary to satisfy the query can be found as index keys, then the table probe is not required. The query uses index-only access. Avoiding the table probe can be an important savings for a query. The I/O associated with a table probe is typically the more expensive synchronous random I/O.

Visual Explain icon:



Radix index scan

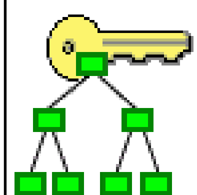
A radix index scan operation is used to retrieve the rows from a table in a keyed sequence. Like a table scan, all the rows in the index are sequentially processed, but the resulting row numbers are sequenced based upon the key columns.

The sequenced rows can be used by the optimizer to satisfy a portion of the query request (such as ordering or grouping). They can also be used to provide faster throughput by performing selection against the index keys rather than all the rows in the table. Since the index I/Os only contain keys, typically more rows can be paged into memory in one I/O than rows in a table with many columns.

Table 4. Radix index scan attributes

Data access method	**Radix index scan**
Description	Sequentially scan and process all the keys associated with the index. Any selection is applied to every key value of the index before a table row
Advantages	• Only those index entries that match any selection continue to be processed • Potential to extract all the data from the index key values, thus eliminating the need for a Table Probe • Returns the rows back in a sequence based upon the keys of the index
Considerations	Generally requires a Table Probe to be performed to extract any remaining columns required to satisfy the query. Can perform poorly when many rows are selected because of the random I/O associated with the Table Probe.
Likely to be used	• When asking for or expecting only a few rows to be returned from the index • When sequencing the rows is required for the query (for example, ordering or grouping) • When the selection columns cannot be matched against the leading key columns of the index

Example SQL statement	`CREATE INDEX X1 ON Employee (LastName, WorkDept)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`ORDER BY LastName`
	`OPTIMIZE FOR 30 ROWS`

Table 4. Radix index scan attributes (continued)	
Data access method	**Radix index scan**
Messages indicating use	• Optimizer Debug: `CPI4328 -- Access path of file X1 was used by query.` • PRTSQLINF: `SQL4008 -- Index X1 used for table 1.`
SMP parallel enabled	Yes
Also referred to as	Index Scan Index Scan, Preload Index Scan, Distinct Index Scan Distinct, Preload Index Scan, Key Selection
Visual Explain icon	

Related reference

Effects of the ALWCPYDTA parameter on database performance
Some complex queries can perform better by using a sort or hashing method to evaluate the query instead of using or creating an index.

Radix index probe

A radix index probe operation is used to retrieve the rows from a table in a keyed sequence. The main difference between the radix index probe and the scan is that the rows returned are first identified by a probe operation to subset them.

The optimizer attempts to match the columns used for some or all the selection against the leading keys of the index. It then rewrites the selection into a series of ranges that can be used to probe directly into the index key values. Only those keys from the series of ranges are paged into main memory.

The resulting row numbers generated by the probe can then be further processed by any remaining selection against the index keys or a table probe operation. This method provides for quick access to only the rows of the index that satisfy the selection.

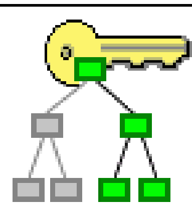
The main function of a radix index probe is to provide quick selection against the index keys. In addition, the row sequencing can be used to satisfy other portions of the query, such as ordering or grouping. Since the index I/Os are only for rows that match the probe selection, no extraneous processing is performed on rows that do not match. This savings in I/Os against rows that are not a part of the result set is one of the primary advantages for this operation.

Table 5. Radix index probe attributes	
Data access method	**Radix index probe**
Description	The index is quickly probed based upon the selection criteria that were rewritten into a series of ranges. Only those keys that satisfy the selection are used to generate a table row number.

Table 5. Radix index probe attributes (continued)	
Data access method	**Radix index probe**
Advantages	• Only those index entries that match any selection continue to be processed • Provides quick access to the selected rows • Potential to extract all the data from the index key values, thus eliminating the need for a Table Probe • Returns the rows back in a sequence based upon the keys of the index
Considerations	Generally requires a Table Probe to be performed to extract any remaining columns required to satisfy the query. Can perform poorly when many rows are selected because of the random I/O associated with the Table Probe.
Likely to be used	• When asking for or expecting only a few rows to be returned from the index • When sequencing the rows is required the query (for example, ordering or grouping) • When the selection columns match the leading key columns of the index

Example SQL statement	`CREATE INDEX X1 ON Employee (LastName, WorkDept)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`AND LastName IN ('Smith', 'Jones', 'Peterson')`
	`OPTIMIZE FOR ALL ROWS`

Messages indicating use	• Optimizer Debug:
	`CPI4328 -- Access path of file X1 was used by query.`
	• PRTSQLINF:
	`SQL4008 -- Index X1 used for table 1.`
	`SQL4011 -- Index scan-key row positioning used on table 1.`

SMP parallel enabled	Yes
Also referred to as	Index Probe Index Probe, Preload Index Probe, Distinct Index Probe Distinct, Preload Index Probe, Key Positioning Index Scan, Key Row Positioning
Visual Explain icon	

The following example illustrates a query where the optimizer might choose the radix index probe access method:

```
CREATE INDEX X1 ON Employee (LastName, WorkDept)

SELECT * FROM Employee
WHERE WorkDept BETWEEN 'A01' AND 'E01'
AND LastName IN ('Smith', 'Jones', 'Peterson')
OPTIMIZE FOR ALL ROWS
```

In this example, index X1 is used to position to the first index entry that matches the selection built over both columns LastName and WorkDept. The selection is rewritten into a series of ranges that match all the leading key columns used from the index X1. The probe is then based upon the composite concatenated values for all the leading keys. The pseudo-SQL for this rewritten SQL might look as follows:

```
SELECT * FROM X1
WHERE X1.LeadingKeys BETWEEN 'JonesA01' AND 'JonesE01'
   OR X1.LeadingKeys BETWEEN 'PetersonA01' AND 'PetersonE01'
   OR X1.LeadingKeys BETWEEN 'SmithA01' AND 'SmithE01'
```

All the key entries that satisfy the probe operation are used to generate a row number for the table associated with the index (for example, Employee). The row number is used by a Table Probe operation to perform random I/O on the table to produce the results for the query. This processing continues until all the rows that satisfy the index probe operation have been processed. In this example, all the index entries processed and rows retrieved met the index probe criteria.

Additional selection might be added that cannot use an index probe, such as selection against columns which are not leading key columns of the index. Then the optimizer performs an index scan operation within the range of probed values. This process still allows for selection to be performed before the Table Probe operation.

Related concepts

Nested loop join implementation
Db2 for i provides a **nested loop** join method. For this method, the processing of the tables in the join are ordered. This order is called the **join order**. The first table in the final join order is called the **primary table**. The other tables are called **secondary tables**. Each join table position is called a **dial**.

Related reference

Effects of the ALWCPYDTA parameter on database performance
Some complex queries can perform better by using a sort or hashing method to evaluate the query instead of using or creating an index.

Encoded vector index

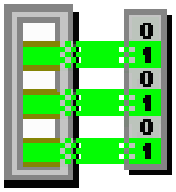
An encoded vector index is a permanent object that provides access to a table. This access is done by assigning codes to distinct key values and then representing those values in a vector.

The size of the vector matches the number of rows in the underlying table. Each vector entry represents the table row number in the same position. The codes generated to represent the distinct key values can be 1 byte, 2 bytes, or 4 bytes in length. The key length depends upon the number of distinct values that need to be represented in the vector. Because of their compact size and relative simplicity, the EVI can be used to process large amounts of data efficiently.

An encoded vector index is used to represent the values stored in a table. However, the index itself cannot be used to directly gain access to the table. Instead, the encoded vector index can only be used to generate either a temporary row number list or a temporary row number bitmap. These temporary objects can then be used with a table probe to specify the rows in the table that the query needs to process.

The main difference in the table probe using an encoded vector index vs. a radix index is that the I/O paging can be asynchronous. The I/O can now be scheduled more efficiently to take advantage of groups of selected rows. Large portions of the table can be skipped over where no rows are selected.

Visual explain icon:



Related concepts

Encoded vector indexes
An encoded vector index (EVI) is used to provide fast data access in decision support and query reporting environments.

EVI maintenance
There are unique challenges to maintaining EVIs. The following table shows a progression of how EVIs are maintained, the conditions under which EVIs are most effective, and where EVIs are least effective, based on the EVI maintenance characteristics.

Encoded vector index probe

The encoded vector index (EVI) is quickly probed based upon the selection criteria that were rewritten into a series of ranges. It produces either a temporary row number list or bitmap.

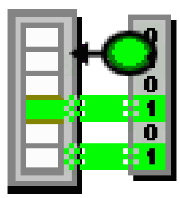
Table 6. Encoded vector index probe attributes

Data access method	Encoded vector index probe
Description	The encoded vector index (EVI) is quickly probed based upon the selection criteria that were rewritten into a series of ranges. It produces either a temporary row number list or bitmap.
Advantages	• Only those index entries that match any selection continue to be processed • Provides quick access to the selected rows • Returns the row numbers in ascending sequence so that the Table Probe can be more aggressive in pre-fetching the rows for its operation
Considerations	EVIs are usually built over a single key. The more distinct the column is and the higher the overflow percentage, the less advantageous the encoded vector index becomes. EVIs always require a Table Probe to be performed on the result of the EVI probe operation.
Likely to be used	• When the selection columns match the leading key columns of the index • When an encoded vector index exists and savings in reduced I/O against the table justifies the extra cost. This cost includes probing the EVI and fully populating the temporary row number list.

Example SQL statement	`CREATE ENCODED VECTOR INDEX EVI1 ON Employee (WorkDept)`
	`CREATE ENCODED VECTOR INDEX EVI2 ON Employee (Salary)`
	`CREATE ENCODED VECTOR INDEX EVI3 ON Employee (Job)`
	
	`SELECT *`
	`FROM Employee`
	`WHERE WorkDept = 'E01' AND Job = 'CLERK'`
	`AND Salary = 5000`
	`OPTIMIZE FOR 99999 ROWS`

Table 6. Encoded vector index probe attributes (continued)	
Data access method	**Encoded vector index probe**

Messages indicating use	• Optimizer Debug:
	`CPI4329 -- Arrival sequence was used for file EMPLOYEE.`
	`CPI4338 -- 3 Access path(s) used for bitmap processing of file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4010 -- Table scan access for table 1.`
	`SQL4032 -- Index EVI1 used for bitmap processing of table 1.`
	`SQL4032 -- Index EVI2 used for bitmap processing of table 1.`
	`SQL4032 -- Index EVI3 used for bitmap processing of table 1.`

SMP parallel enabled	Yes
Also referred to as	Encoded Vector Index Probe, Preload
Visual Explain icon	

Using the example above, the optimizer chooses to create a temporary row number bitmap for each of the encoded vector indexes used by this query. Each bitmap only identifies those rows that match the selection on the key columns for that index.

These temporary row number bitmaps are then merged together to determine the intersection of the rows selected from each index. This intersection is used to form a final temporary row number bitmap used to help schedule the I/O paging against the table for the selected rows.

The optimizer might choose to perform an index probe with a binary radix tree index if an index existed over all three columns. The implementation choice is probably decided by the number of rows to be returned and the anticipated cost of the I/O associated with each plan.

If few rows are returned, the optimizer probably chooses the binary radix tree index and performs the random I/O against the table. However, selecting more rows causes the optimizer to use the EVIs, because of the savings from the more efficiently scheduled I/O against the table.

Encoded vector index index-only access

The encoded vector index can also be used for index-only access.

The EVI can be used for more than generating a bitmap or row number list to provide an asynchronous I/O map to the desired table rows. The EVI can also be used by two index-only access methods that can be applied specific to the symbol table itself. These two index-only access methods are the EVI symbol table scan and the EVI symbol table probe.

These two methods can be used with GROUP BY or DISTINCT queries that can be satisfied by the symbol table. This symbol table-only access can be further employed in aggregate queries by adding INCLUDE values to the encoded vector index.

The following information is a summary of the symbol table-only scan and probe access methods.

Use the following links to learn in-depth information.

Related concepts

Encoded vector indexes
An encoded vector index (EVI) is used to provide fast data access in decision support and query reporting environments.

How the EVI works
EVIs work in different ways for costing and implementation.

Related reference

Index grouping implementation
There are two primary ways to implement grouping using an index: Ordered grouping and pre-summarized processing.

Encoded vector index symbol table scan

An encoded vector index symbol table scan operation is used to retrieve the entries from the symbol table portion of the index.

All entries (symbols) in the symbol table are sequentially scanned if a scan is chosen. The symbol table can be used by the optimizer to satisfy GROUP BY or DISTINCT portions of a query request.

Selection is applied to every entry in the symbol table. The selection must be applied to the symbol table keys unless the EVI was created as a sparse index, with a WHERE clause. In that case, a portion of the selection is applied as the symbol table is built and maintained. The query request must include matching predicates to use the sparse EVI.

All entries are retrieved directly from the symbol table portion of the index without any access to the vector portion of the index. There is also no access to the records in the associated table over which the EVI is built.

Encoded vector index INCLUDE aggregates

To enhance the ability of the EVI symbol table to provide aggregate answers, the symbol table can be created to contain additional INCLUDE values. These are ready-made numeric aggregate results, such as SUM, COUNT, AVG, or VARIANCE values requested over non-key data. These aggregates are specified using the INCLUDE keyword on the CREATE ENCODED VECTOR INDEX request.

These included aggregates are maintained in real time as rows are inserted, updated, or deleted from the corresponding table. The symbol table maintains these additional aggregate values in addendum to the EVI keys for each symbol table entry. Because these are numeric results and finite in size, the symbol table is still a desirable compact size.

These included aggregates are over non-key columns in the table where the grouping is over the corresponding EVI symbol table defined keys. The aggregate can be over a single column or a derivation.

Table 7. Encoded vector index symbol table scan attributes

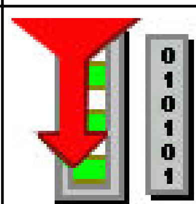
Data access method	Encoded vector index symbol table scan
Description	Sequentially scan and process all the symbol table entries associated with the index. When there is selection (WHERE clause), it is applied to every entry in the symbol table. An exception is made in the case of a sparse EVI, where the selection is applied as the index is created and maintained. Selected entries are retrieved directly without any access to the vector or the associated table.

Table 7. Encoded vector index symbol table scan attributes (continued)	
Data access method	**Encoded vector index symbol table scan**
Advantages	• Pre-summarized results are readily available • Only processes the unique values in the symbol table, avoiding processing table records. • Extract all the data from the index unique key values or INCLUDE values, thus eliminating the need for a Table Probe or vector scan. • With INCLUDE, provides ready-made numeric aggregates, eliminating the need to access corresponding table rows to perform the aggregation
Considerations	Dramatic performance improvement for grouping queries where the resulting number of groups is relatively small compared to the number of records in the underlying table. Can perform poorly when there are many groups involved such that the symbol table is large. Poor performance is even more likely if a large portion of the symbol table has been put into the overflow area. Dramatic performance improvement for grouping queries when the aggregate is specified as an INCLUDE value of the symbol table.
Likely to be used	• When asking for GROUP BY, DISTINCT, COUNT, or COUNT DISTINCT from a single table and the referenced columns are in the key definition. • When the number of unique values in the columns of the key definition is small relative to the number of records in the underlying table. • When there is no selection (WHERE clause) within the query or the selection does not reduce the result set much. • When the symbol table key satisfies the GROUP BY, and requested aggregates, like SUM or COUNT, are specified as INCLUDE values. • when the query is run with commitment control *NONE or *CHG.

Table 7. Encoded vector index symbol table scan attributes (continued)	
Data access method	**Encoded vector index symbol table scan**

Example SQL statement	`CREATE ENCODED VECTOR INDEX EVI1 ON Sales (Region)`
	Example 1
	`SELECT Region, count(*)`
	`FROM Sales`
	`GROUP BY Region`
	`OPTIMIZE FOR ALL ROWS`
	Example 2
	`SELECT DISTINCT Region`
	`FROM Sales`
	`OPTIMIZE FOR ALL ROWS`
	Example 3
	`SELECT COUNT(DISTINCT Region)`
	`FROM Sales`
	Example 4 uses the INCLUDE option. The sums of revenue and cost of goods per sales region is maintained in real time.
	`CREATE ENCODED VECTOR INDEX EVI2 ON Sales(Region)`
	`INCLUDE(SUM(Revenue), SUM(CostOfGoods))`
	
	`SELECT Region, SUM(Revenue), SUM(CostOfGoods)`
	`FROM Sales`
	`GROUP BY Region`

Messages indicating use	• Optimizer Debug:
	`CPI4328 -- Access path of file EVI1 was used by query.`
	• PRTSQLINF:
	`SQL4008 -- Index EVI1 used for table 1.SQL4010`

Also referred to as	Encoded Vector Index Table Scan, Preload
Visual Explain icon	

Related concepts

Encoded vector indexes
An encoded vector index (EVI) is used to provide fast data access in decision support and query reporting environments.

How the EVI works
EVIs work in different ways for costing and implementation.

Related reference

Index grouping implementation

There are two primary ways to implement grouping using an index: Ordered grouping and pre-summarized processing.

Related information

SQL INCLUDE statement

Encoded vector index symbol table probe

An encoded vector index symbol table probe operation is used to retrieve entries from the symbol table portion of the index. Scanning the entire symbol table is not necessary.

The symbol table can be used by the optimizer to satisfy GROUP BY or DISTINCT portions of a query request.

The optimizer attempts to match the columns used for some or all the selection against the leading keys of the EVI index. It then rewrites the selection into a series of ranges that can be used to probe directly into the symbol table. Only those symbol table pages from the series of ranges are paged into main memory.

The resulting symbol table entries generated by the probe operation can then be further processed by any remaining selection against EVI keys. This strategy provides for quick access to only the entries of the symbol table that satisfy the selection.

Like an encoded vector symbol table scan, a symbol table probe can return ready-made aggregate results if INCLUDE is specified when the EVI is created.

All entries are retrieved directly from the symbol table portion of the index without any access to the vector portion of the index. In addition, it is unnecessary to access the records in the associated table over which the EVI is built.

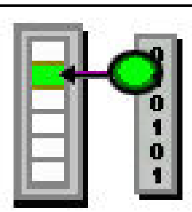
Table 8. Encoded vector index symbol table probe attributes

Data access method	**Encoded vector index symbol table probe**
Description	
Advantages	Probe the symbol table entries associated with the index. When there is selection (WHERE clause), it is applied to every entry in the symbol table that meets the probe criteria. If there are sparse EVIs, the selection is applied as the EVI is created and maintained. Selected entries are retrieved directly without any access to the vector or the associated table.
Considerations	• Pre-summarized results are readily available • Only processes the unique values in the symbol table, avoiding processing table records. • Extracts all the data from the index unique key values or include values, or both, thus eliminating the need for a table probe or vector scan • With INCLUDE, provides ready-made numeric aggregates, eliminating the need to access corresponding table rows to perform the aggregation

Table 8. Encoded vector index symbol table probe attributes (continued)	
Data access method	**Encoded vector index symbol table probe**
Likely to be used	• When asking for GROUP BY, DISTINCT, COUNT, or COUNT DISTINCT from a single table and the referenced columns are in the key definition. • When the number of unique values in the columns of the key definition is small relative to the number of records in the underlying table. • When there is selection (WHERE clause) that reduces the selection from the Symbol Table and the WHERE clause involves leading, probable keys. • When the symbol table key satisfies the GROUP BY and the WHERE clause reduces selection to the leading keys, and aggregates are specified as INCLUDE values. • When the query is run with commitment control *NONE or *CHG.

Example SQL statement	`CREATE ENCODED VECTOR INDEX EVI1 ON Sales (Region)`
	Example 1
	`SELECT Region, COUNT(*)`
	`FROM Sales`
	`WHERE Region in ('Quebec', 'Manitoba')`
	`GROUP BY Region`
	`OPTIMIZE FOR ALL ROWS`
	Example 2
	`CREATE ENCODED VECTOR INDEX  EVI2 ON Sales(Region)`
	`INCLUDE(SUM(Revenue), SUM(CostOfGoods))`
	
	`SELECT Region, SUM(Revenue), SUM(CostOfGoods)`
	`FROM Sales`
	`WHERE Region = 'PACIFIC'`
	`GROUP BY Region`

Messages indicating use	• Optimizer Debug:
	`CPI4328 -- Access path of file EVI1 was used by query.`
	• PRTSQLINF:
	`SQL4008 -- Index EVI1 used for table 1.SQL4010`

Also referred to as	Encoded Vector Index Table Probe, Preload
Visual Explain icon	

Related concepts
Encoded vector indexes

An encoded vector index (EVI) is used to provide fast data access in decision support and query reporting environments.

How the EVI works
EVIs work in different ways for costing and implementation.

Related reference

Index grouping implementation
There are two primary ways to implement grouping using an index: Ordered grouping and pre-summarized processing.

Related information

SQL INCLUDE statement

Temporary objects and access methods

Temporary objects are created by the optimizer in order to process a query. In general, these temporary objects are internal objects and cannot be accessed by a user.

Table 9. Temporary object data access methods

Temporary create objects	Scan operations	Probe operations
Temporary hash table	Hash table scan	Hash table probe
Temporary sorted list	Sorted list scan	Sorted list probe
Temporary distinct sorted list	Sorted list scan	N/A
Temporary list	List scan	N/A
Temporary values list	Values list scan	N/A
Temporary row number list	Row number list scan	Row number list probe
Temporary bitmap	Bitmap scan	Bitmap probe
Temporary index	Temporary index scan	Temporary index probe
Temporary buffer	Buffer scan	N/A
Queue	N/A	N/A
Array unnest temporary table	Temporary table scan	N/A

Temporary hash table

The temporary hash table is a temporary object that allows the optimizer to collate the rows based upon a column or set of columns. The hash table can be either scanned or probed by the optimizer to satisfy different operations of the query.

A temporary hash table is an efficient data structure because the rows are organized for quick and easy retrieval after population has occurred. The hash table remains resident within main memory to avoid any I/Os associated with either the scan or probe against the temporary object. The optimizer determines the optimal hash table size based on the number of unique column combinations used as keys for the creation.

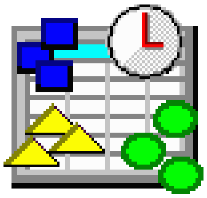
Additionally the hash table can be populated with all the necessary columns to satisfy any further processing. This population avoids any random I/Os associated with a table probe operation.

However, the optimizer can selectively include columns in the hash table when the calculated size exceeds the memory pool storage available for the query. In these cases, a table probe operation is required to recollect the missing columns from the hash table before the selected rows can be processed.

The optimizer also can populate the hash table with distinct values. If the query contains grouping or distinct processing, then all the rows with the same key value are not required in the hash table. The rows are still collated, but the distinct processing is performed during the population of the hash table itself. This method allows a simple scan on the result in order to complete the grouping or distinct operation.

A temporary hash table is an internal data structure and can only be created by the database manager

Visual explain icon:



Hash table scan

During a hash table scan operation, the entire temporary hash table is scanned and all the entries contained within the hash table are processed.

The optimizer considers a hash table scan when the data values need to be collated together, but sequencing of the data is not required. A hash table scan allows the optimizer to generate a plan that takes advantage of any non-join selection while creating the temporary hash table.

An additional benefit is that the temporary hash table data structure will typically cause the table data to remain resident within main memory after creation. Resident table data reduces paging on the subsequent hash table scan operation.

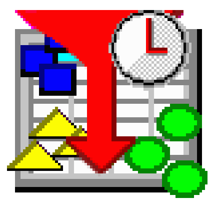
Table 10. Hash table scan attributes

Data access method	Hash table scan
Description	Read all the entries in a temporary hash table. The hash table can perform distinct processing to eliminate duplicates. Or the temporary hash table can collate all the rows with the same value together.
Advantages	• Reduces the random I/O to the table associated with longer running queries that might otherwise use an index to collate the data • Selection can be performed before generating the hash table to subset the number of rows in the temporary object
Considerations	Used for distinct or group by processing. Can perform poorly when the entire hash table does not stay resident in memory as it is being processed.
Likely to be used	• When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA) • When the data is required to be collated based upon a column or columns for distinct or grouping

Example SQL statement	`SELECT COUNT(*), FirstNme FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`GROUP BY FirstNme`

Table 10. Hash table scan attributes (continued)	
Data access method	**Hash table scan**

Messages indicating use	There are multiple ways in which a hash scan can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a hash scan was used.
	• Optimizer Debug:
	`CPI4329 -- Arrival sequence was used for file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4010 -- Table scan access for table 1.`
	`SQL4029 -- Hashing algorithm used to process the grouping.`

SMP parallel enabled	Yes
Also referred to as	Hash Scan, Preload Hash Table Scan Distinct Hash Table Scan Distinct, Preload
Visual Explain icon	

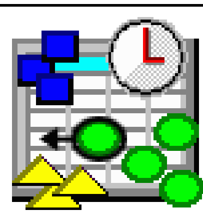
Hash table probe

A hash table probe operation is used to retrieve rows from a temporary hash table based upon a probe lookup operation.

The optimizer initially identifies the keys of the temporary hash table from the join criteria specified in the query. When the hash table is probed, the values used to probe into the hash table are extracted from the join-from criteria specified in the selection.

These values are sent through the same hashing algorithm used to populate the temporary hash table. They determine if any rows have a matching equal value. All the matching join rows are then returned to be further processed by the query.

Table 11. Hash table probe attributes	
Data access method	**Hash table probe**
Description	The temporary hash table is quickly probed based upon the join criteria.
Advantages	• Provides quick access to the selected rows that match probe criteria • Reduces the random I/O to the table associated with longer running queries that use an index to collate the data • Selection can be performed before generating the hash table to subset the number of rows in the temporary object
Considerations	Used to process equal join criteria. Can perform poorly when the entire hash table does not stay resident in memory as it is being processed.

<table>
<caption>Table 11. Hash table probe attributes (continued)</caption>
<tr><th>Data access method</th><th>Hash table probe</th></tr>
<tr><td>Likely to be used</td><td>When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA)
When the data is required to be collated based upon a column or columns for join processing
The join criteria was specified using an equals (=) operator</td></tr>
<tr><td>Example SQL statement</td><td><pre>SELET * FROM Employee XXX, Department YYY
WHERE XXX.WorkDept = YYY.DeptNbr
OPTIMIZE FOR ALL ROWS</pre></td></tr>
<tr><td>Messages indicating use</td><td>There are multiple ways in which a hash probe can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a hash probe was used.
Optimizer Debug:<pre>CPI4327 -- File EMPLOYEE processed in join
           position 1.
CPI4327 -- File DEPARTMENT processed in join
           position 2.</pre>
PRTSQLINF:<pre>SQL4007 -- Query implementation for join
           position 1 table 1.
SQL4010 -- Table scan access for table 1.
SQL4007 -- Query implementation for join
           position 2 table 2.
SQL4010 -- Table scan access for table 2.</pre></td></tr>
<tr><td>SMP parallel enabled</td><td>Yes</td></tr>
<tr><td>Also referred to as</td><td>Hash Table Probe, Preload
Hash Table Probe Distinct
Hash Table Probe Distinct, Preload</td></tr>
<tr><td>Visual Explain icon</td><td></td></tr>
</table>

The hash table probe access method is considered when determining the implementation for a secondary table of a join. The hash table is created with the key columns that match the equal selection or join criteria for the underlying table.

The hash table probe allows the optimizer to choose the most efficient implementation in selecting rows from the underlying table, without regard for join criteria. This single pass through the underlying table can now use a table scan or existing index to select the rows needed for the hash table population.

Since hash tables are constructed so that most of the hash table remains resident within main memory, the I/O associated with a hash probe is minimal. Additionally, if the hash table was populated with all necessary columns from the underlying table, no additional table probe is required to finish processing this table. This method causes further I/O savings.

Related concepts

Nested loop join implementation

Db2 for i provides a **nested loop** join method. For this method, the processing of the tables in the join are ordered. This order is called the **join order**. The first table in the final join order is called the **primary table**. The other tables are called **secondary tables**. Each join table position is called a **dial**.

Temporary sorted list

The temporary sorted list is a temporary object that allows the optimizer to sequence rows based upon a column or set of columns. The sorted list can be either scanned or probed by the optimizer to satisfy different operations of the query.

A temporary sorted list is a data structure where the rows are organized for quick and easy retrieval after population has occurred. During population, the rows are copied into the temporary object and then a second pass is made through the temporary object to perform the sort.

In order to optimize the creation of this temporary object, minimal data movement is performed while the sort is processed. It is not as efficient to probe a temporary sorted list as it is to probe a temporary hash table.

Additionally, the sorted list can be populated with all the necessary columns to satisfy any further processing. This population avoids any random I/Os associated with a table probe operation.

However, the optimizer can selectively include columns in the sorted list when the calculated size exceeds the memory pool storage available for this query. In those cases, a table probe operation is required to recollect the missing columns from the sorted list before the selected rows can be processed.

A temporary sorted list is an internal data structure and can only be created by the database manager.

Visual explain icon:

Sorted list scan

During a sorted list scan operation, the entire temporary sorted list is scanned and all the entries contained within the sorted list are processed.

A sorted list scan is considered when the data values need to be sequenced. A sorted list scan allows the optimizer to generate a plan that can take advantage of any non-join selection while creating the temporary sorted list.

An additional benefit is that the data structure will usually cause the table data within the sorted list to remain resident within main memory after creation. This resident data reduces paging on the subsequent sorted list scan operation.

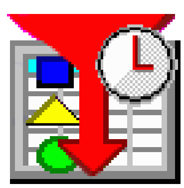
Table 12. Sorted list scan attributes

Data access method	Sorted list scan
Description	Read all the entries in a temporary sorted list. The sorted list can perform distinct processing to eliminate duplicate values or take advantage of the temporary sorted list to sequence all the rows.
Advantages	• Reduces the random I/O to the table associated with longer running queries that would otherwise use an index to sequence the data. • Selection can be performed prior to generating the sorted list to subset the number of rows in the temporary object

Table 12. Sorted list scan attributes (continued)	
Data access method	**Sorted list scan**
Considerations	Used to process ordering or distinct processing. Can perform poorly when the entire sorted list does not stay resident in memory as it is being populated and processed.
Likely to be used	• When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA) • When the data is required to be ordered based upon a column or columns for ordering or distinct processing

Example SQL statement	`CREATE INDEX X1 ON Employee (LastName, WorkDept)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`ORDER BY FirstNme`
	`OPTIMZE FOR ALL ROWS`

Messages indicating use	There are multiple ways in which a sorted list scan can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a sorted list scan was used.
	• Optimizer Debug:
	`CPI4328 -- Access path of file X1 was used by query.`
	`CPI4325 -- Temporary result file built for query.`
	• PRTSQLINF:
	`SQL4008 -- Index X1 used for table 1.`
	`SQL4002 -- Reusable ODP sort used.`

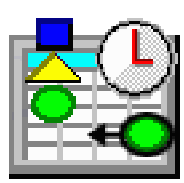
SMP parallel enabled	No
Also referred to as	Sorted List Scan, Preload Sorted List Scan Distinct Sorted List Scan Distinct, Preload
Visual Explain icon	

Sorted list probe

A sorted list probe operation is used to retrieve rows from a temporary sorted list based upon a probe lookup operation.

The optimizer initially identifies the temporary sorted list keys from the join criteria specified in the query. The values used to probe into the temporary sorted list are extracted from the join-from criteria specified in the selection. Those values are used to position within the sorted list in order to determine if any rows have a matching value. All the matching join rows are then returned to be further processed by the query.

Table 13. Sorted list probe attributes	
Data access method	**Sorted list probe**
Description	The temporary sorted list is quickly probed based upon the join criteria.

<table>
<caption>Table 13. Sorted list probe attributes (continued)</caption>
<tr><th>Data access method</th><th>Sorted list probe</th></tr>
<tr><td>Advantages</td><td>Provides quick access to the selected rows that match probe criteria
Reduces the random I/O to the table associated with longer running queries that otherwise use an index to collate the data
Selection can be performed before generating the sorted list to subset the number of rows in the temporary object</td></tr>
<tr><td>Considerations</td><td>Used to process non-equal join criteria. Can perform poorly when the entire sorted list does not stay resident in memory as it is being populated and processed.</td></tr>
<tr><td>Likely to be used</td><td>When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA)
When the data is required to be collated based upon a column or columns for join processing
The join criteria was specified using a non-equals operator</td></tr>
<tr><td>Example SQL statement</td><td><pre>SELECT * FROM Employee XXX, Department YYY
WHERE XXX.WorkDept > YYY.DeptNo
OPTIMIZE FOR ALL ROWS</pre></td></tr>
<tr><td>Messages indicating use</td><td>There are multiple ways in which a sorted list probe can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a sorted list probe was used.
Optimizer Debug:<pre>CPI4327 -- File EMPLOYEE processed in join position 1.
CPI4327 -- File DEPARTMENT processed in join
           position 2.</pre>
PRTSQLINF:<pre>SQL4007 -- Query implementation for join
           position 1 table 1.
SQL4010 -- Table scan access for table 1.
SQL4007 -- Query implementation for join
           position 2 table 2.
SQL4010 -- Table scan access for table 2.</pre></td></tr>
<tr><td>SMP parallel enabled</td><td>Yes</td></tr>
<tr><td>Also referred to as</td><td>Sorted List Probe, Preload
Sorted List Probe Distinct
Sorted List Probe Distinct, Preload</td></tr>
<tr><td>Visual Explain icon</td><td></td></tr>
</table>

The sorted list probe access method is considered when determining the implementation for a secondary table of a join. The sorted list is created with the key columns that match the non-equal join criteria for the underlying table. The optimizer chooses the most efficient implementation to select the rows from the underlying table without regard to any join criteria. This single pass through the underlying table can use a Table Scan or an existing index to select the rows needed to populate the sorted list.

Since sorted lists are constructed so that most of the temporary object remains resident within main memory, the sorted list I/O is minimal. If the sorted list was populated with all necessary table columns, no additional Table Probe is required to finish processing the table, causing further I/O savings.

Related concepts

Nested loop join implementation
Db2 for i provides a **nested loop** join method. For this method, the processing of the tables in the join are ordered. This order is called the **join order**. The first table in the final join order is called the **primary table**. The other tables are called **secondary tables**. Each join table position is called a **dial**.

Temporary distinct sorted list

A temporary distinct sorted list combines the features of the temporary hash table and the temporary sorted list.

Like the hash table, the temporary distinct sorted list allows the optimizer to collate the rows based on a column or set of columns. Like the sorted list, the temporary distinct sorted list also allows the optimizer to sequence the rows.

A temporary distinct sorted list contains a hash table data structure set up for efficient access to aggregate rows during population. In addition, a binary tree data structure is maintained over the hash table data structure so that the data can be accessed in sequence. The sorted aspect of the data structure allows for the efficient computation of super-aggregate rows in SQL statements that contain GROUP BY ROLLUP.

A temporary sorted aggregate hash table is an internal data structure and can only be created by the database manager.

Visual explain icon:

Sorted list scan

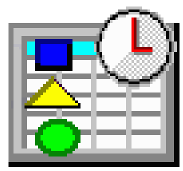
During the sorted list scan, the entire temporary distinct sorted list is scanned and all the entries contained within the temporary are processed.

The optimizer uses the sorted list scan when the data values need to be aggregated and sequenced. The optimizer generates this plan that can take advantage of any non-join selection while creating the temporary distinct sorted list. The data structure of the temporary distinct sorted list will typically cause the table data to remain resident within main memory after creation. This memory-resident data reduces paging on the subsequent sorted list scan.

Table 14. Sorted list scan attributes

Data access method	Sorted list scan
Description	Reads all the entries in a temporary distinct sorted list
Advantages	• Allows efficient computation of ROLLUP super-aggregate rows. • Reduces the random I/O to the table associated with longer running queries that might otherwise use an index to collate the data. • Selection can be performed before generating the distinct sorted list to subset the number of rows in the temporary object.

Table 14. Sorted list scan attributes (continued)

Data access method	Sorted list scan
Considerations	Used for GROUP BY ROLLUP processing, Can perform poorly when the entire temporary object does not stay resident in memory as it is being processed.
Likely to be used	• When the use of temporary results is allowed in the query environmental parameter (ALWCPYDTA) • When a GROUP BY ROLLUP is in the SQL statement
Messages indicating use	N/A
SMP parallel enabled	Yes
Also referred to as	N/A
Visual Explain icon	

Temporary list

The temporary list is a temporary object that allows the optimizer to store intermediate results of a query. The list is an unsorted data structure that is used to simplify the operation of the query. Since the list does not have any keys, the rows within the list can only be retrieved by a sequential scan operation.

The temporary list can be used for various reasons, some of which include an overly complex view or derived table, Symmetric Multiprocessing (SMP) or to prevent a portion of the query from being processed multiple times.

A temporary list is an internal data structure and can only be created by the database manager.

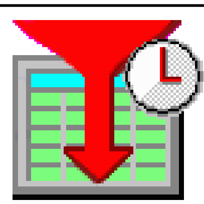
Visual explain icon:

List scan

The list scan operation is used when a portion of the query is processed multiple times, but no key columns can be identified. In these cases, that portion of the query is processed once and its results are stored within the temporary list. The list can then be scanned for only those rows that satisfy any selection or processing contained within the temporary object.

Table 15. List scan attributes

Data access method	List scan
Description	Sequentially scan and process all the rows in the temporary list.

<table>
<caption>Table 15. List scan attributes (continued)</caption>
<tr><th>Data access method</th><th>List scan</th></tr>
<tr><td>Advantages</td><td>The temporary list and list scan can be used by the optimizer to minimize repetition of an operation or to simplify the optimizer logic flow.
Selection can be performed before generating the list to subset the number of rows in the temporary object.</td></tr>
<tr><td>Considerations</td><td>Used to prevent portions of the query from being processed multiple times when no key columns are required to satisfy the request.</td></tr>
<tr><td>Likely to be used</td><td>When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA).
When DB2 symmetric multiprocessing is used for the query.</td></tr>
<tr><td>Example SQL statement</td><td><pre>SELECT * FROM Employee XXX, Department YYY
WHERE XXX.LastName IN ('Smith', 'Jones', 'Peterson')
AND YYY.DeptNo BETWEEN 'A01' AND 'E01'
OPTIMIZE FOR ALL ROWS</pre></td></tr>
<tr><td>Messages indicating use</td><td>There are multiple ways in which a list scan can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a list scan was used.
Optimizer Debug:<pre>CPI4325 -- Temporary result file built for query.
CPI4327 -- File EMPLOYEE processed in join
           position 1.
CPI4327 -- File DEPARTMENT processed in join
           position 2.</pre>
PRTSQLINF:<pre>SQL4007 -- Query implementation for join
           position 1 table 1.
SQL4010 -- Table scan access for table 1.
SQL4007 -- Query implementation for join
           position 2 table 2.
SQL4001 -- Temporary result created
SQL4010 -- Table scan access for table 2.</pre></td></tr>
<tr><td>SMP parallel enabled</td><td>Yes</td></tr>
<tr><td>Also referred to as</td><td>List Scan, Preload</td></tr>
<tr><td>Visual Explain icon</td><td></td></tr>
</table>

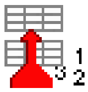
Using the example above, the optimizer chose to create a temporary list to store the selected rows from the DEPARTMENT table. Since there is no join criteria, a Cartesian product join is performed between the two tables. To prevent the join from scanning all the rows of the DEPARTMENT table for each join possibility, the selection against the DEPARTMENT table is performed once. The results are stored in the temporary list. The temporary list is then scanned for the Cartesian product join.

Temporary values list
The temporary values list allows the optimizer to store rows of data specified in a VALUES clause of a SELECT or CREATE VIEW statement.

The list is an unsorted data structure that is used to simplify the operation of the query. Since the list does not have any keys, the rows within the list can only be retrieved by a sequential scan operation.

A temporary values list is an internal data structure and can only be created by the database manager.

Visual explain icon:



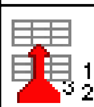
Values list scan
During a values list scan operation, the entire temporary values list is scanned and all the rows of data are processed.

Table 16. Values list scan attributes

Data access method	Values list scan
Description	Sequentially scan and process all the rows of data in the temporary values list.
Advantages	The temporary values list and values list scan can be used by the optimizer to simplify the optimizer logic flow.
Likely to be used	When a VALUES clause is specified in the from-clause of an SQL fullselect

Example SQL statement	`SELECT EMPNO, 'empprojact'`
	`FROM EMPPROJACT`
	`WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')`
	`UNION`
	`VALUES ('NEWAAA', 'new'), ('NEWBBB', 'new')`

Messages indicating use	There are multiple ways in which a values list scan can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a values list scan was used.
	• Optimizer Debug:
	`CPI4329 -- Arrival sequence was used for file *VALUES.`
	• PRTSQLINF:
	`SQL4010 -- Table scan access for table 1.`

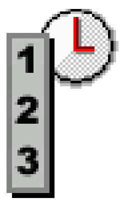
SMP parallel enabled	Yes
Also referred to as	Values List, Preload
Visual Explain icon	

Temporary row number list
The temporary row number list is a temporary object that allows the optimizer to sequence rows based upon their row address (their row number). The row number list can be either scanned or probed by the optimizer to satisfy different operations of the query.

A temporary row number list is a data structure where the rows are organized for quick and efficient retrieval. The row number list only contains the row number for the associated row. Since no table data is present, a table probe operation is typically associated with it in order to retrieve the underlying table data. Because the row numbers are sorted, the random I/O associated with the table probe operation is performed more efficiently. The database manager performs pre-fetch or look-ahead logic to determine if multiple rows are located on adjacent pages. If so, the table probe requests a larger I/O to bring the rows into main memory more efficiently.

A temporary row number list is an internal data structure and can only be created by the database manager.

Visual explain icon:



Row number list scan
The entire temporary row number list is scanned and all the row addresses contained within the row number list are processed. The optimizer considers this plan when there is an applicable encoded vector index or if the index probe or scan random I/O can be reduced. The random I/O can be reduced by first preprocessing and sorting the row numbers associated with the Table Probe.

The use of a row number list scan allows the optimizer to generate a plan that can take advantage of multiple indexes to match up to different portions of the query.

An additional benefit is that the data structure of the temporary row number list guarantees that the row numbers are sorted. It closely mirrors the row number layout of the table data, ensuring that the table paging never visits the same page of data twice. This results in increased I/O savings for the query.

A row number list scan is identical to a bitmap scan operation. The only difference is that the list scan is over a list of row addresses while the bitmap scan is over a bitmap representing the addresses.

Table 17. Row number list scan

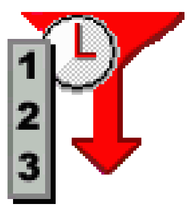
Data access method	**Row number list scan**
Description	Sequentially scan and process all the row numbers in the temporary row number list. The sorted row numbers can be merged with other temporary row number lists or can be used as input into a Table Probe operation.
Advantages	• The temporary row number list only contains address, no data, so the temporary can be efficiently scanned within memory. • The row numbers contained within the temporary object are sorted to provide efficient I/O processing to access the underlying table. • Selection is performed as the row number list is generated to subset the number of rows in the temporary object.
Considerations	Since the row number list contains only the addresses of the selected rows in the table, a separate Table Probe fetches the table rows.
Likely to be used	• When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA). • When the cost of sorting of the row number is justified by the more efficient I/O that can be performed during the Table Probe operation. • When multiple indexes over the same table need to be combined in order to minimize the number of selected rows.

Table 17. Row number list scan (continued)

Data access method	Row number list scan

Example SQL statement	`CREATE INDEX X1 ON Employee (WorkDept)`
	`CREATE ENCODED VECTOR INDEX EVI2 ON`
	`     Employee (Salary)`
	`CREATE ENCODED VECTOR INDEX EVI3 ON`
	`     Employee (Job)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept = 'E01' AND Job = 'CLERK'`
	`AND Salary = 5000`
	`OPTIMIZE FOR 99999 ROWS`

Messages indicating use	There are multiple ways in which a row number list scan can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a row number list scan was used.
	
	• Optimizer Debug:
	
	`CPI4329 -- Arrival sequence was used for file`
	`           EMPLOYEE.`
	`CPI4338 -- 3 Access path(s) used for bitmap`
	`           processing of file EMPLOYEE.`
	
	• PRTSQLINF:
	
	`SQL4010 -- Table scan access for table 1.`
	`SQL4032 -- Index X1 used for bitmap`
	`           processing of table 1.`
	`SQL4032 -- Index EVI2 used for bitmap`
	`           processing of table 1.`
	`SQL4032 -- Index EVI3 used for bitmap`
	`           processing of table 1.`

SMP parallel enabled	Yes
Also referred to as	Row Number List Scan, Preload
Visual Explain icon	

Using the example above, the optimizer created a temporary row number list for each of the indexes used by this query. These indexes included a radix index and two encoded vector indexes. Each index row number list was scanned and merged into a final composite row number list representing the intersection of all the index row number lists. The final row number list is then used by the Table Probe to determine which rows are selected and processed for the query results.

Row number list probe

A row number list probe is used to test row numbers generated by a separate operation against the selected rows of a temporary row number list. The row numbers can be generated by any operation that constructs a row number for a table. That row number is then used to probe into a temporary row number list to determine if it matches the selection used to generate the list.

The use of a row number list probe operation allows the optimizer to generate a plan that can take advantage of any sequencing provided by an index, but still use the row number list to perform additional selection before any Table probe operations.

A row number list probe is identical to a bitmap probe operation. The only difference is that the list probe is over a list of row addresses while the bitmap probe is over a bitmap representing the addresses.

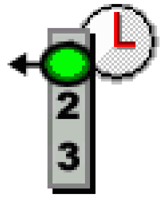
Table 18. Row number list probe

Data access method	**Row number list probe**
Description	The temporary row number list is quickly probed based upon the row number generated by a separate operation.
Advantages	• The temporary row number list only contains a row address, no data, so the temporary can be efficiently probed within memory. • The row numbers represented within the row number list are sorted to provide efficient lookup processing to test the underlying table. • Selection is performed as the row number list is generated to subset the number of selected rows in the temporary object.
Considerations	Since the row number list contains only the addresses of the selected rows in the table, a separate Table Probe fetches the table rows.
Likely to be used	• When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA). • When the cost of creating and probing the row number list is justified by reducing the number of Table Probe operations that must be performed. • When multiple indexes over the same table need to be combined in order to minimize the number of selected rows.

Example SQL statement	`CREATE INDEX X1 ON Employee (WorkDept)`
	`CREATE ENCODED VECTOR INDEX EVI2 ON`
	`     Employee (Salary)`
	`CREATE ENCODED VECTOR INDEX EVI3 ON`
	`     Employee (Job)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept = 'E01' AND Job = 'CLERK'`
	` AND Salary = 5000`
	`ORDER BY WorkDept`

Messages indicating use	There are multiple ways in which a row number list probe can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates a row number list probe was used.
	• Optimizer Debug:
	`CPI4328 -- Access path of file X1 was used by query.`
	`CPI4338 -- 2 Access path(s) used for bitmap`
	`           processing of file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4008 -- Index X1 used for table 1.`
	`SQL4011 -- Index scan-key row positioning`
	`           used on table 1.`
	`SQL4032 -- Index EVI2 used for bitmap`
	`           processing of table 1.`
	`SQL4032 -- Index EVI3 used for bitmap`
	`           processing of table 1.`

SMP parallel enabled	Yes
Also referred to as	Row Number List Probe, Preload

Table 18. Row number list probe (continued)	
Data access method	**Row number list probe**
Visual Explain icon	

Using the example above, the optimizer created a temporary row number list for each of the encoded vector indexes. Additionally, an index probe operation was performed against the radix index X1 to satisfy the ordering requirement. Since the ORDER BY requires that the resulting rows be sequenced by the `WorkDept` column, the row number list cannot be scanned for the selected rows.

However, the temporary row number list can be probed using a row address extracted from the index X1 used to satisfy the ordering. By probing the list with the row address extracted from the index probe, the sequencing of the keys in the index X1 is preserved. The row can still be tested against the selected rows within the row number list.

Temporary bitmap

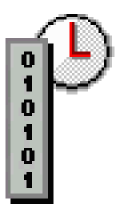
The temporary bitmap is a temporary object that allows the optimizer to sequence rows based upon their row address (their row number). The bitmap can be either scanned or probed by the optimizer to satisfy different operations of the query.

A temporary bitmap is a data structure that uses a bitmap to represent all the row numbers for a table. Since each row is represented by a separate bit, all the rows within a table can be represented in a fairly condensed form. When a row is selected, the bit within the bitmap that corresponds to the selected row is set on. After the temporary bitmap is populated, all the selected rows can be retrieved in a sorted manner for quick and efficient retrieval. The temporary bitmap only represents the row number for the associated selected rows.

No table data is present within the temporary bitmap. A table probe operation is typically associated with the bitmap in order to retrieve the underlying table data. Because the bitmap is by definition sorted, the random I/O associated with the table probe operation can be performed more efficiently. The database manager performs pre-fetch or look-ahead logic to determine if multiple rows are located on adjacent pages. If so, the table probe requests a larger I/O to bring the rows into main memory more efficiently.

A temporary bitmap is an internal data structure and can only be created by the database manager.

Visual explain icon:



Bitmap scan

During a bitmap scan operation, the entire temporary bitmap is scanned and all the row addresses contained within the bitmap are processed. The optimizer considers this plan when there is an applicable encoded vector index or if the index probe or scan random I/O can be reduced. The random I/O can be reduced by first preprocessing and sorting the row numbers associated with the Table Probe.

The use of a bitmap scan allows the optimizer to generate a plan that can take advantage of multiple indexes to match up to different portions of the query.

An additional benefit is that the data structure of the temporary bitmap guarantees that the row numbers are sorted. It closely mirrors the row number layout of the table data, ensuring that the table paging never visits the same page of data twice. This results in increased I/O savings for the query.

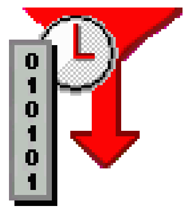
A bitmap scan is identical to a row number list scan operation. The only difference is that the list scan is over a list of row addresses while the bitmap scan is over a bitmap representing the addresses.

Table 19. Bitmap scan attributes

Data access method	Bitmap scan attributes
Description	Sequentially scan and process all the row numbers in the temporary bitmap. The sorted row numbers can be merged with other temporary bitmaps or can be used as input into a Table Probe operation.
Advantages	• The temporary bitmap only contains a reference to a row address, no data, so the temporary can be efficiently scanned within memory. • The row numbers represented within the temporary object are sorted to provide efficient I/O processing to access the underlying table. • Selection is performed as the bitmap is generated to subset the number of selected rows in the temporary object.
Considerations	Since the bitmap contains only the addresses of the selected rows in the table, a separate Table Probe fetches the table rows.
Likely to be used	• When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA). • When the cost of sorting of the row numbers is justified by the more efficient I/O that can be performed during the Table Probe operation. • When multiple indexes over the same table need to be combined in order to minimize the number of selected rows.

Example SQL statement	`CREATE INDEX X1 ON Employee (WorkDept)`
	`CREATE ENCODED VECTOR INDEX EVI2 ON`
	`     Employee (Salary)`
	`CREATE ENCODED VECTOR INDEX EVI3 ON`
	`     Employee (Job)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept = 'E01' AND Job = 'CLERK'`
	`AND Salary = 5000`
	`OPTIMIZE FOR 99999 ROWS`

Messages indicating use	There are multiple ways in which a bitmap scan can be indicated through the messages. The messages in this example illustrate how the Classic Query Engine indicates a bitmap scan was used.
	• Optimizer Debug:
	`CPI4329 -- Arrival sequence was used for file`
	`           EMPLOYEE.`
	`CPI4338 -- 3 Access path(s) used for bitmap`
	`           processing of file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4010 -- Table scan access for table 1.`
	`SQL4032 -- Index X1 used for bitmap`
	`           processing of table 1.`
	`SQL4032 -- Index EVI2 used for bitmap`
	`           processing of table 1.`
	`SQL4032 -- Index EVI3 used for bitmap`
	`           processing of table 1.`

Table 19. Bitmap scan attributes (continued)	
Data access method	**Bitmap scan attributes**
SMP parallel enabled	Yes
Also referred to as	Bitmap Scan, Preload Row Number Bitmap Scan Row Number Bitmap Scan, Preload Skip Sequential Scan
Visual Explain icon	

Using the example above, the optimizer created a temporary bitmap for each of the indexes used by this query. These indexes included a radix index and two encoded vector indexes. Each index temporary bitmap was scanned and merged into a final composite bitmap representing the intersection of all the index temporary bitmaps. The final bitmap is then used by the Table Probe operation to determine which rows are selected and processed for the query results.

Bitmap probe

A bitmap probe operation is used to test row numbers generated by a separate operation against the selected rows of a temporary bitmap. The row numbers can be generated by any operation that constructs a row number for a table. That row number is then used to probe into a temporary bitmap to determine if it matches the selection used to generate the bitmap.

The use of a bitmap probe operation allows the optimizer to generate a plan that can take advantage of any sequencing provided by an index, but still use the bitmap to perform additional selection before any Table Probe operations.

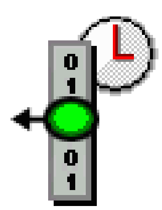
A bitmap probe is identical to a row number list probe operation. The only difference is that the list probe is over a list of row addresses while the bitmap probe is over a bitmap representing the addresses.

Table 20. Bitmap probe attributes	
Data access method	**Bitmap probe attributes**
Description	The temporary bitmap is quickly probed based upon the row number generated by a separate operation.
Advantages	• The temporary bitmap only contains a reference to a row address, no data, so the temporary can be efficiently probed within memory. • The row numbers represented within the bitmap are sorted to provide efficient lookup processing to test the underlying table. • Selection is performed as the bitmap is generated to subset the number of selected rows in the temporary object.
Considerations	Since the bitmap contains only the addresses of the selected rows in the table, a separate Table Probe fetches the table rows.

Table 20. Bitmap probe attributes (continued)	
Data access method	**Bitmap probe attributes**
Likely to be used	• When the use of temporary results is allowed by the query environmental parameter (ALWCPYDTA). • When the cost of creating and probing the bitmap is justified by reducing the number of Table Probe operations that must be performed. • When multiple indexes over the same table need to be combined in order to minimize the number of selected rows.

Example SQL statement	`CREATE INDEX X1 ON Employee (WorkDept)`
	`CREATE ENCODED VECTOR INDEX EVI2 ON`
	`     Employee (Salary)`
	`CREATE ENCODED VECTOR INDEX EVI3 ON`
	`     Employee (Job)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept = 'E01' AND Job = 'CLERK'`
	`AND Salary = 5000`
	`ORDER BY WorkDept`

Messages indicating use	There are multiple ways in which a bitmap probe can be indicated through the messages. The messages in this example illustrate how the Classic Query Engine indicates a bitmap probe was used.
	• Optimizer Debug:
	`CPI4328 -- Access path of file X1 was used by query.`
	`CPI4338 -- 2 Access path(s) used for bitmap`
	`           processing of file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4008 -- Index X1 used for table 1.`
	`SQL4011 -- Index scan-key row positioning`
	`           used on table 1.`
	`SQL4032 -- Index EVI2 used for bitmap`
	`           processing of table 1.`
	`SQL4032 -- Index EVI3 used for bitmap`
	`           processing of table 1.`

SMP parallel enabled	Yes
Also referred to as	Bitmap Probe, Preload Row Number Bitmap Probe Row Number Bitmap Probe, Preload
Visual Explain icon	

Using the example above, the optimizer created a temporary bitmap for each of the encoded vector indexes. Additionally, an index probe operation was performed against the radix index X1 to satisfy the ordering requirement. Since the ORDER BY requires that the resulting rows be sequenced by the `WorkDept` column, the bitmap cannot be scanned for the selected rows.

However, the temporary bitmap can be probed using a row address extracted from the index X1 used to satisfy the ordering. By probing the bitmap with the row address extracted from the index probe, the sequencing of the keys in the index X1 is preserved. The row can still be tested against the selected rows within the bitmap.

Temporary index

A temporary index is a temporary object that allows the optimizer to create and use a radix index for a specific query. The temporary index has all the same attributes and benefits as a radix index created through the CREATE INDEX SQL statement or **Create Logical File (CRTLF)** CL command.

Additionally, the temporary index is optimized for use by the optimizer to satisfy a specific query request. This optimization includes setting the logical page size and applying any selection to the index to speed up its use after creation.

The temporary index can be used to satisfy various query requests:

- Ordering
- Grouping/Distinct
- Joins
- Record selection

Generally a temporary index is a more expensive temporary object to create than other temporary objects. It can be populated by a table scan, or by one or more index scans or probes. The optimizer considers all the methods available when determining which method to use to produce the rows for the index creation. This process is like the costing and selection of the other temporary objects used by the optimizer.

One significant advantage of the temporary index over other temporary objects is that it is the only temporary object maintained if the underlying table changes. The temporary index is identical to a radix index in that any inserts or updates against the table are reflected immediately through normal index maintenance.

SQE usage of temporary indexes is different from CQE usage in that SQE allows reuse. References to temporary indexes created and used by the SQE optimizer are kept in the system Plan Cache. A temporary index is saved for reuse by other instances of the same query or other instances of the same query running in a different job. It is also saved for potential reuse by a different query that can benefit from the use of the same temporary index.

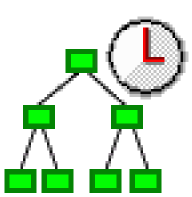
By default, an SQE temporary index persists until the Plan Cache entry for the last referencing query plan is removed. With the SQE Plan Cache auto sizing capability, there is the potential for SQE temporary indexes to persist longer. You can control this behavior by setting the CACHE_RESULTS QAQQINI value. The default for this INI value allows the optimizer to keep temporary indexes around for reuse.

Changing the INI value to '*JOB' prevents the temporary index from being saved in the Plan Cache; the index does not survive a hard close. The *JOB option causes the SQE optimizer use of temporary indexes to behave more like the CQE optimizer. The temporary index has a shorter life, but is still shared as long as there are active queries using it. This behavior can be desirable in cases where there is concern about increased maintenance costs for temporary indexes that persist for reuse.

A SQE temporary index can also be used as a source of statistics.

A temporary index is an internal data structure and can only be created by the database manager.

Visual explain icon:



Temporary index scan
A temporary index scan operation is identical to the index scan operation that is performed upon the permanent radix index. It is still used to retrieve the rows from a table in a keyed sequence; however, the temporary index object must first be created. All the rows in the index are sequentially processed, but the resulting row numbers are sequenced based upon the key columns.

The sequenced rows can be used by the optimizer to satisfy a portion of the query request (such as ordering or grouping).

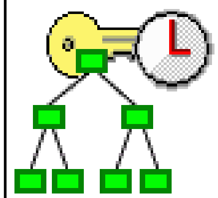
Table 21. Temporary index scan attributes

Data access method	Temporary index scan
Description	Sequentially scan and process all the keys associated with the temporary index.
Advantages	• Potential to extract all the data from the index key values, thus eliminating the need for a Table Probe • Returns the rows back in a sequence based upon the keys of the index
Considerations	Generally requires a Table Probe to be performed to extract any remaining columns required to satisfy the query. Can perform poorly when many rows are selected because of the random I/O associated with the Table Probe.
Likely to be used	• When sequencing the rows is required for the query (for example, ordering or grouping) • When the selection columns cannot be matched against the leading key columns of the index • When the overhead cost associated with the creation of the temporary index can be justified against other alternative methods to implement this query

Example SQL statement	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`ORDER BY LastName`
	`OPTIMIZE FOR ALL ROWS`

Messages indicating use	• Optimizer Debug:
	`CPI4321 -- Access path built for file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4009 -- Index created for table 1.`

SMP parallel enabled	Yes
Also referred to as	Index Scan Index Scan, Preload Index Scan, Distinct Index Scan Distinct, Preload Index Scan, Key Selection

Table 21. Temporary index scan attributes (continued)	
Data access method	**Temporary index scan**
Visual Explain icon	

Using the example above, the optimizer chose to create a temporary index to sequence the rows based upon the LastName column. A temporary index scan might then be performed to satisfy the ORDER BY clause in this query.

The optimizer determines where the selection against the `WorkDept` column best belongs. It can be performed as the temporary index itself is being created or it can be performed as a part of the temporary index scan. Adding the selection to the temporary index creation has the possibility of making the open data path (ODP) for this query non-reusable. This ODP reuse is considered when determining how selection is performed.

Temporary index probe

A temporary index probe operation is identical to the index probe operation that is performed on the permanent radix index. Its main function is to provide quick access against the index keys of the temporary index. However, it can still be used to retrieve the rows from a table in a keyed sequence.

The temporary index is used by the optimizer to satisfy the join portion of the query request.

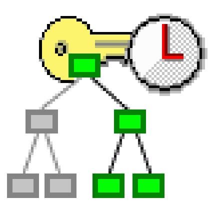
Table 22. Temporary index probe attributes	
Data access method	**Temporary index probe**
Description	The index is quickly probed based upon the selection criteria that were rewritten into a series of ranges. Only those keys that satisfy the selection is used to generate a table row number.
Advantages	• Only those index entries that match any selection continue to be processed. Provides quick access to the selected rows • Potential to extract all the data from the index key values, thus eliminating the need for a Table Probe • Returns the rows back in a sequence based upon the keys of the index
Considerations	Generally requires a Table Probe to be performed to extract any remaining columns required to satisfy the query. Can perform poorly when many rows are selected because of the random I/O associated with the Table Probe.
Likely to be used	• When the ability to probe the rows required for the query (for example, joins) exists • When the selection columns cannot be matched against the leading key columns of the index • When the overhead cost associated with the creation of the temporary index can be justified against other alternative methods to implement this query

Table 22. Temporary index probe attributes (continued)

Data access method	Temporary index probe

Example SQL statement	`SELET * FROM Employee XXX, Department YYY`
	`WHERE XXX.WorkDept = YYY.DeptNo`
	`OPTIMIZE FOR ALL ROWS`

Messages indicating use	There are multiple ways in which a temporary index probe can be indicated through the messages. The messages in this example illustrate one example of how the Classic Query Engine indicates a temporary index probe was used.
	• Optimizer Debug:
	`CPI4321 -- Access path built for file DEPARTMENT.`
	`CPI4327 -- File EMPLOYEE processed in join`
	`           position 1.`
	`CPI4326 -- File DEPARTMENT processed in join`
	`           position 2.`
	• PRTSQLINF:
	`SQL4007 -- Query implementation for join`
	`           position 1 table 1.`
	`SQL4010 -- Table scan access for table 1.`
	`SQL4007 -- Query implementation for join`
	`           position 2 table 2.`
	`SQL4009 -- Index created for table 2.`

SMP parallel enabled	Yes
Also referred to as	Index Probe Index Probe, Preload Index Probe, Distinct Index Probe Distinct, Preload Index Probe, Key Selection
Visual Explain icon	

Using the example above, the optimizer chose to create a temporary index over the DeptNo column to help satisfy the join requirement against the DEPARTMENT table. A temporary index probe was then performed against the temporary index to process the join criteria between the two tables. In this particular case, there was no additional selection that might be applied against the DEPARTMENT table while the temporary index was being created.

Temporary buffer

The temporary buffer is a temporary object that is used to help facilitate operations such as parallelism. It is an unsorted data structure that is used to store intermediate rows of a query. The difference between a temporary buffer and a temporary list is that the buffer does not need to be fully populated before its results are processed.

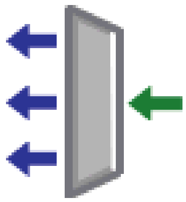
The temporary buffer acts as a serialization point between parallel and non-parallel portions of a query. The operations used to populate the buffer cannot be performed in parallel, whereas the operations that fetch rows from the buffer can be performed in parallel.

The temporary buffer is required for SQE because the index scan and index probe operations are not SMP parallel-enabled for this engine. Unlike CQE, which performs these index operations in parallel, SQE does not subdivide the index operation work to take full advantage of parallel processing.

The buffer is used to allow a query to be processed under parallelism by serializing access to the index operations. Any remaining work within the query is processed in parallel.

A temporary buffer is an internal data structure and can only be created by the database manager.

Visual explain icon:



Buffer scan

The buffer scan is used when a query is processed using DB2 Symmetric Multiprocessing, yet a portion of the query is unable to be parallel processed. The buffer scan acts as a gateway to control access to rows between the parallel enabled portions of the query and the non-parallel portions.

Multiple threads can be used to fetch the selected rows from the buffer, allowing the query to perform any remaining processing in parallel. However, the buffer is populated in a non-parallel manner.

A buffer scan operation is identical to the list scan operation that is performed upon the temporary list object. The main difference is that a buffer does not need to be fully populated before the start of the scan operation. A temporary list requires that the list is fully populated before fetching any rows.

Table 23. Buffer scan attributes

Data access method	**Buffer scan**
Description	Sequentially scan and process all the rows in the temporary buffer. Enables SMP parallelism to be performed over a non-parallel portion of the query.
Advantages	• The temporary buffer can be used to enable parallelism over a portion of a query that is non-parallel • The temporary buffer does not need to be fully populated in order to start fetching rows
Considerations	Used to prevent portions of the query from being processed multiple times when no key columns are required to satisfy the request.
Likely to be used	• When the query is attempting to take advantage of DB2 Symmetric Multiprocessing • When a portion of the query cannot be performed in parallel (for example, index scan or index probe)

Example SQL statement	`CHGQRYA DEGREE(*OPTIMIZE)`
	`CREATE INDEX X1 ON`
	`     Employee (LastName, WorkDept)`
	
	`SELECT * FROM Employee`
	`WHERE WorkDept BETWEEN 'A01' AND 'E01'`
	`AND LastName IN ('Smith', 'Jones', 'Peterson')`
	`OPTIMIZE FOR ALL ROWS`

Table 23. Buffer scan attributes (continued)

Data access method	Buffer scan

Messages indicating use	• Optimizer Debug:
	`CPI4328 -- Access path of file X1 was used by query.`
	`CPI4330 -- 8 tasks used for parallel index scan`
	`of file EMPLOYEE.`
	• PRTSQLINF:
	`SQL4027 -- Access plan was saved with DB2`
	`SMP installed on the system.`
	`SQL4008 -- Index X1 used for table 1.`
	`SQL4011 -- Index scan-key row positioning`
	`used on table 1.`
	`SQL4030 -- 8 tasks specified for parallel scan`
	`on table 1.`

SMP parallel enabled	Yes
Also referred to as	Not applicable
Visual Explain icon	

Using the example above, the optimizer chose to use the existing index X1 to perform an index probe operation against the table. In order to speed up the remaining Table Probe operation for this query, DB2 Symmetric Multiprocessing is used to perform the random probe into the table. Since the index probe is not SMP parallel-enabled for SQE, it is placed within a temporary buffer to control access to the selected index entries.

Queue

The Queue is a temporary object that the optimizer uses to feed recursion by putting data values needed for the recursion on it. This data typically includes those values used on the recursive join predicate, and other recursive data accumulated or manipulated during the recursive process.

The Queue has two operations allowed:

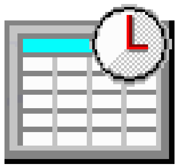
- Enqueue: puts data on the queue
- Dequeue: takes data off the queue

A queue is an efficient data structure because it contains only the data needed to feed the recursion or directly modified by the recursion process. Its size is managed by the optimizer.

Unlike other temporary objects created by the optimizer, the queue is not populated all at once by the underlying query node tree. It is a real-time temporary holding area for values feeding the recursion. In this regard, a queue is not considered temporary, as it does not prevent the query from running if ALWCPYDTA(*NO) was specified. The data can flow from the query at the same time the recursive values are inserted into the queue and used to retrieve additional join rows.

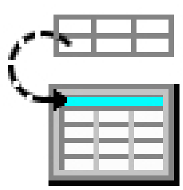
A queue is an internal data structure and can only be created by the database manager.

Visual explain icon:



Enqueue
During an enqueue operation, an entry is put on the queue. The entry contains key values used by the recursive join predicates or data manipulated as a part of the recursion process. The optimizer always supplies an enqueue operation to collect the required recursive data on the query node directly above the Union All.

Table 24. Enqueue Attributes

Data Access Method	Enqueue
Description	Places an entry on the queue needed to cause further recursion
Advantages	• Required as a source for the recursion. Only enqueues required values for the recursion process. Each entry has short life span, until it is dequeued. • Each entry on the queue can seed multiple iterative fullselects that are recursive from the same RCTE or view.
Likely to be used	A required access method for recursive queries
Example SQL statement	`WITH RPL (PART, SUBPART, QUANTITY) AS (SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY FROM PARTLIST ROOT WHERE ROOT.PART = '01' UNION ALL SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY FROM RPL PARENT, PARTLIST CHILD WHERE PARENT.SUBPART = CHILD.PART) SELECT DISTINCT PART, SUBPART, QUANTITY FROM RPL`
Messages indicating use	There are no explicit messages that indicate the use of an enqueue
SMP parallel enabled	Yes
Also referred to as	Not applicable
Visual Explain icon	

Use the CYCLE option in the definition of the recursive query if the data reflecting the parent-child relationship could be cyclic, causing an infinite recursion loop. CYCLE prevents already visited recursive key values from being put on the queue again for a given set of related (ancestry chain) rows.

Use the SEARCH option in the definition of the recursive query to return the results of the recursion in the specified parent-child hierarchical ordering. The search choices are Depth or Breadth first. Depth first means that all the descendents of each immediate child are returned before the next child is returned. Breadth first means that each child is returned before their children are returned.

SEARCH requires not only the specification of the relationship keys, the columns which make up the parent-child relationship, and the search type of Depth or Breadth. It also requires an ORDER BY clause in the main query on the provided sequence column in order to fully implement the specified ordering.

Dequeue

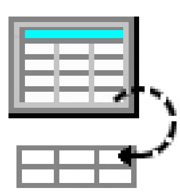
During a dequeue operation, an entry is taken off the queue. Those values specified by recursive reference are fed back in to the recursive join process.

The optimizer always supplies a corresponding enqueue, dequeue pair of operations for each recursive common table expression or recursive view in the specifying query. Recursion ends when there are no more entries to pull off the queue.

Table 25. Dequeue Attributes

Data Access Method	Dequeue
Description	Removes an entry off the queue. Minimally, provides one side of the recursive join predicate that feeds the recursive join and other data values that are manipulated through the recursive process. The dequeue operation is always on the left side of the inner join with constraint, where the right side is the target child rows.
Advantages	• Provides quick access to recursive values • Allows for post selection of local predicate on recursive data values
Likely to be used	• A required access method for recursive queries • A single dequeued value can feed the recursion of multiple iterative fullselects that reference the same RCTE or view

Example SQL statement	`WITH RPL (PART, SUBPART, QUANTITY) AS`
	`( SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY`
	`FROM PARTLIST ROOT`
	`WHERE ROOT.PART = '01'`
	`UNION ALL`
	`SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY`
	`FROM RPL PARENT, PARTLIST CHILD`
	`WHERE PARENT.SUBPART = CHILD.PART`
	`)`
	`SELECT DISTINCT PART, SUBPART, QUANTITY`
	`FROM RPL`

Messages indicating use	There are no explicit messages that indicate the use of the dequeue operation.
SMP parallel enabled	Yes
Also referred to as	Not applicable
Visual Explain icon	

Array unnest temporary table

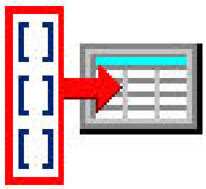
The array unnest temporary table is a temporary object that holds the output of an UNNEST of an array or a list of arrays. It can be viewed vertically, with each column of array values having the same format. The temporary table contains one or more arrays specified by the user in an UNNEST clause of a SELECT statement.

UNNEST creates a temporary table with the arrays specified as columns in the table. If more than one array is specified, the first array provides the first column in the result table. The second array provides the second column, and so on.

The arrays might be of different lengths. Shorter arrays are primed with nulls to match the length of the longest array in the list.

If WITH ORDINALITY is specified, an extra counter column of type BIGINT is appended to the temporary table. The ordinality column contains the index position of the elements in the arrays.

The array unnest temporary table is an internal data structure and can only be created by the database manager.

Visual explain icon: 

Related reference

QAQQINI query options

There are different options available for parameters in the QAQQINI file.

Related information

Array support in SQL procedures

Debugging an SQL routine

table-reference

Array unnest temporary table scan

During an array unnest temporary table scan operation, the temporary table is processed one row at a time.

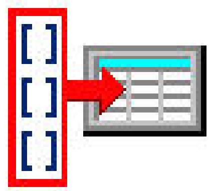
Table 26. Array unnest temporary table scan operation

Data access method	Array unnest temporary table scan
Description	Sequentially scan and process all the rows of data in the unnest temporary table.
Advantages	The array unnest temporary table and temporary table scan can be used to simplify the logic flow of the optimizer for processing arrays.
Likely to be used	When an UNNEST clause is specified in the from-clause of an SQL fullselect.

Example SQL statement	`CREATE PROCEDURE processCustomers()`
	`BEGIN`
	`DECLARE ids INTARRAY;`
	`DECLARE names STRINGARRAY;`
	`set ids = ARRAY[5,6,7];`
	`set names = ARRAY['Ann', 'Bob', 'Sue'];`
	`INSERT INTO customerTable(id, name, order)`
	`(SELECT Customers.id, Customers.name, Customers.order`
	`FROM UNNEST(ids, names) WITH ORDINALITY`
	`AS Customers(id, name, order) );`
	`END`
	
	`CALL processCustomers()`

Table 26. Array unnest temporary table scan operation (continued)	
Data access method	**Array unnest temporary table scan**

Messages indicating use	There are multiple ways in which an array unnest temporary table scan can be indicated through the messages. The messages in this example illustrate how the SQL Query Engine indicates an array unnest temporary table scan was used.
	• Optimizer Debug:
	`CPI4329 -- Arrival sequence was used for file *UNNEST_1.`
	• PRTSQLINF:
	`SQL4010 -- Table scan access for table 1.`

SMP parallel enabled	Yes
Also referred to as	
Visual Explain icon	

Objects processed in parallel

The DB2 Symmetric multiprocessing feature provides the optimizer with additional methods for retrieving data that include parallel processing. Symmetrical multiprocessing is a form of parallelism achieved on a single system where multiple CPU and I/O processors sharing memory and disk work simultaneously toward a single result.

This parallel processing means that the database manager can have more than one (or all) of the system processors working on a single query simultaneously. The performance of a CPU-bound query can be improved with this feature on multiple-processor systems by distributing the processor load across more than one processor.

The preceding tables indicate what data access methods are enabled to take advantage of the DB2 Symmetric Multiprocessing feature. An important thing to note, however, is that the parallel implementation differs for both the SQL Query Engine and the Classic Query Engine.

Processing requirements

Parallelism requires that SMP parallel processing must be enabled by one of the following methods:

- System value QQRYDEGREE
- Query option file
- DEGREE parameter on the **Change Query Attributes (CHGQRYA)** command
- SQL SET CURRENT DEGREE statement

Once parallelism has been enabled, a set of database system tasks or threads is created at system startup for use by the database manager. The database manager uses the tasks to process and retrieve data from different disk devices. Since these tasks can be run on multiple processors simultaneously, the elapsed time of a query can be reduced. Even though the tasks do much of the parallel I/O and CPU processing, the I/O and CPU resource accounting is transferred to the application job. The summarized I/O and CPU resources for this type of application continue to be accurately displayed by the **Work with Active Jobs (WRKACTJOB)** command.

The job must be run in a shared storage pool with the *CALC paging option, as this method causes more efficient use of active memory.

Related concepts

Nested loop join implementation
Db2 for i provides a **nested loop** join method. For this method, the processing of the tables in the join are ordered. This order is called the **join order**. The first table in the final join order is called the **primary table**. The other tables are called **secondary tables**. Each join table position is called a **dial**.

Related reference

Changing the attributes of your queries
You can modify different types of query attributes for a job with the **`Change Query Attributes (CHGQRYA)`** CL command. You can also use the System i Navigator Change Query Attributes interface.

Related information

SET CURRENT DEGREE statement

Performance system values: Parallel processing for queries and indexes

Adjusting performance automatically

Work with Active Jobs (WRKACTJOB) command

Change Query Attributes (CHGQRYA) command

DB2 Symmetric Multiprocessing

Spreading data automatically

Db2 for i automatically spreads the data across the disk devices available in the auxiliary storage pool (ASP) where the data is allocated. This process ensures that the data is spread without user intervention.

The spreading allows the database manager to easily process the blocks of rows on different disk devices in parallel. Even though Db2 for i spreads data across disk devices within an ASP, sometimes the allocation of the data extents (contiguous sets of data) might not be spread evenly. This unevenness occurs when there is uneven allocation of space on the devices, or when a new device is added to the ASP. The allocation of the table data space could be spread again by saving, deleting, and then restoring the table.

Maintaining an even distribution of data across all the disk devices can lead to better throughput on query processing. The number of disk devices used and how the data is spread across them is considered by the optimizer while costing the different plan permutations.

Processing queries: Overview

This overview of the query optimizer provides guidelines for designing queries that perform and use system resources more efficiently.

This overview covers queries that are optimized by the query optimizer and includes interfaces such as SQL, OPNQRYF, APIs (QQQQRY), ODBC, and Query/400 queries. Whether you apply the guidelines, the query results are still correct.

Note: The information in this overview is complex. You might find it helpful to experiment with an IBM i product as you read this information to gain a better understanding of the concepts.

When you understand how Db2 for i processes queries, it is easier to understand the performance impacts of the guidelines discussed in this overview. There are two major components of Db2 for i query processing:

- How the system accesses data.

  These methods are the algorithms that are used to retrieve data from the disk. The methods include index usage and row selection techniques. In addition, parallel access methods are available with the DB2 Symmetric Multiprocessing operating system feature.
- Query optimizer

The query optimizer identifies the valid techniques which can be used to implement the query and selects the most efficient technique.

How the query optimizer makes your queries more efficient

Data manipulation statements such as SELECT specify only what data the user wants, not how to retrieve that data. This path to the data is chosen by the optimizer and stored in the access plan. Understand the techniques employed by the query optimizer for performing this task.

The optimizer is an important part of Db2 for i because the optimizer:

- Makes the key decisions which affect database performance.
- Identifies the techniques which can be used to implement the query.
- Selects the most efficient technique.

General query optimization tips

Here are some tips to help your queries run as fast as possible.

- Create indexes whose leftmost key columns match your selection predicates to help supply the optimizer with selectivity values (key range estimates).
- For join queries, create indexes that match your join columns to help the optimizer determine the average number of matching rows.
- Minimize extraneous mapping by specifying only columns of interest on the query. For example, specify only the columns you need to query on the SQL SELECT statement instead of specifying SELECT *. Also, specify FOR FETCH ONLY if the columns do not need to be updated.
- If your queries often use table scan, use the **`Reorganize Physical File Member (RGZPFM)`** command to remove deleted rows from tables, or the **`Change Physical File (CHGPF)`** REUSEDLT (*YES) command to reuse deleted rows.

Consider using the following options:

- Specify ALWCPYDTA(*OPTIMIZE) to allow the query optimizer to create temporary copies of data so better performance can be obtained. The IBM i Access ODBC driver and Query Management driver always use this mode. If ALWCPYDTA(*YES) is specified, the query optimizer attempts to implement the query without copies of the data, but might create copies if required. If ALWCPYDTA(*NO) is specified, copies of the data are not allowed. If the query optimizer cannot find a plan that does not use a temporary, then the query cannot be run.
- For SQL, use CLOSQLCSR(*ENDJOB) or CLOSQLCSR(*ENDACTGRP) to allow open data paths to remain open for future invocations.
- Specify DLYPRP(*YES) to delay SQL statement validation until an OPEN, EXECUTE, or DESCRIBE statement is run. This option improves performance by eliminating redundant validation.
- Use ALWBLK(*ALLREAD) to allow row blocking for read-only cursors.

Related information

Reorganize Physical File Member (RGZPFM) command

Change Physical File (CHGPF) command

Access plan validation

An access plan is a control structure that describes the actions necessary to satisfy each query request. It contains information about the data and how to extract it. For any query, whenever optimization occurs, the query optimizer develops an optimized plan of how to access the requested data.

To improve performance, an access plan is saved once it is built (see following exceptions), to be available for potentially future runs of the query. However, the optimizer has dynamic replan capability. This means that even if a previously built (and saved) plan is found, the optimizer could rebuild it if a more optimal plan is possible. This process allows for maximum flexibility while still taking advantage of saved plans.

- For dynamic SQL, an access plan is created at prepare or open time. However, optimization uses the host variable values to determine an optimal plan. Therefore, a plan built at prepare time could be rebuilt the first time the query is opened (when the host variable values are present).
- For an IBM i program that contains static embedded SQL, an access plan is initially created at compile time. Again, since optimization uses the host variable values to determine an optimal plan, the compile-time plan could be rebuilt the first time the query is opened.
- For Open Query File (OPNQRYF), an access plan is created but is not saved. A new access plan is created each time the OPNQRYF command is processed.
- For Query/400, an access plan is saved as part of the query definition object.

In all the preceding cases where a plan is saved, including static SQL, dynamic replan can still apply as the queries are run over time.

The access plan is validated when the query is opened. Validation includes the following:

- Verifying that the same tables are referenced in the query as in the access plan. For example, the tables were not deleted and recreated or that the tables resolved by using *LIBL have not changed.
- Verifying that the indexes used to implement the query, still exist.
- Verifying that the table size or predicate selectivity has not changed significantly.
- Verifying that QAQQINI options have not changed.

Single table optimization

At run time, the optimizer chooses an optimal access method for a query by calculating an *implementation cost* based on the current state of the database. The optimizer uses two costs in its decision: an I/O cost and a CPU cost. The goal of the optimizer is to minimize both I/O and CPU cost.

Improved query optimization I/O cost estimates

The time it takes to perform an disk I/O operation can vary according to the connecting infrastructure, the external or internal nature of the media and media type, spinning disk or Solid State Disk. Consequently, the total I/O cost associated with a particular query access method may vary from system to system.

In order to more accurately estimate these costs, the optimizer considers the performance of each disk unit individually. It does this by measuring the time it takes for read operations to complete across a sample of pages across the disk. This analysis is done at each IPL for disks in the system and user ASPs and at vary-on time for independent ASPs. With this information and with the additional knowledge about how database objects are spread across various disk units, the optimizer can make a reasonable estimate about the time it takes to perform I/O against a given database object. This means that no matter where your data resides, and even as it moves around, the optimizer can choose the most efficient plan to execute your queries.

Optimizing Access to each table

The optimizer uses a general set of guidelines to choose the best method for accessing data in each table. The optimizer:

- Determines the default filter factor for each predicate in the selection clause.
- Determines the true filter factor of the predicates by key range estimate when the selection predicates match the index left-most keys, or by available column statistics.
- Determines the cost of table scan processing if an index is not required.
- Determines the cost of creating an index over a table if an index is required. This index is created by performing either a table scan or creating an index-from-index.
- Determines the cost of using a sort routine or hashing method if appropriate.
- Determines the cost of using existing indexes using Index Probe or Index Scan

- Orders the indexes. For SQE, the indexes are ordered in general such that the indexes that access the smallest number of entries are examined first. For CQE, the indexes are ordered from mostly recently created to oldest.
- For each index available, the optimizer does the following:
  - Determines if the index meets the selection criteria.
  - Determines the cost of using the index by estimating the number of I/Os and CPU needed to Index Probe or Index Scan, and possible Table Probes.
  - Compares the cost of using this index with the previous cost (current best).
  - Picks the cheaper one.
  - Continues to search for best index until the optimizer decides to look at no more indexes.

  SQE orders the indexes so that the best indexes are examined first. Once an index is found that is more expensive than the previously chosen best index, the search is ended.

  For CQE, the *time limit* controls how much time the optimizer spends choosing an implementation. The time limit is based on how much time was spent so far and the current best implementation cost found. The idea is to prevent the optimizer from spending more time optimizing the query than it takes to actually execute the query. Dynamic SQL queries are subject to the optimizer time restrictions. Static SQL query optimization time is not limited. For OPNQRYF, if you specify OPTALLAP(*YES), the optimization time is not limited.

  For small tables, the query optimizer spends little time in query optimization. For large tables, the query optimizer considers more indexes. For CQE, the optimizer generally considers five or six indexes for each table of a join before running out of optimization time. Because of this processing, it is normal for the optimizer to spend longer lengths of time analyzing queries against the tables.

- Determines the cost of using a temporary bitmap
  - Order the indexes that can be used for bit mapping. In general the indexes that select the smallest number of entries are examined first.
  - Determine the cost of using this index for bit mapping and the cost of merging this bitmap with any previously generated bitmaps.
  - If the cost of this bitmap plan is cheaper than the previous bitmap plan, continue searching for bitmap plans.
- After examining the possible methods of access the data for the table, the optimizer chooses the best plan from all the plans examined.

Solid State Drives

Solid State Drives (SSDs) offer a number of advantages over traditional hard disk drives (HDDs)

Solid State Drives

Solid State Drives (SSDs) offer a number of advantages over traditional hard disk drives (HDDs). With no seek time or rotational delays, SSDs can deliver substantially better I/O performance than HDDs. Capable of driving tens of thousands of I/O operations per second as opposed to hundreds for HDDs, SSDs break through performance bottlenecks of I/O-bound applications. Applications that require dozens and dozens of “extra” HDDs for performance can meet their I/O performance requirements with far fewer SSDs, resulting in energy, space, and cost savings.

As IBM i has it’s own storage manager and DB2 for i built in, the integration of SSDs on IBM i is a fairly simple task. The functions provided for management of SSDs and adjusting their impact on Applications and Database are very simple and easy to use.

There are three basic methodologies to place data on SSD.

- ASP Balancer – Enhanced for SSDs
- Library and SSD Integration
- DB2 and SSD Integration

To compare and contrast these methodologies see the IBM i white paper:

http://www-03.ibm.com/systems/resources/ssd_ibmi.pdf

To allow you to specify what data should be allocated on SSD, DB2 has provided the capability to specify a "media preference" as an attribute of a database table, partition, or index. It should be noted that this attribute specifies that storage allocations on SSD are preferred, but if no SSD disks are available or if the SSD disks do not have enough space left to allocate the entire object, at least some part of the object will be allocated on traditional disks. See the **UNIT** parameter on **CRTPF** and **CRTLF** or the *media-preference* clause (UNIT SSD) on the **CREATE TABLE**, **DECLARE GLOBAL TEMPORARY TABLE**, **CREATE INDEX**, and **ALTER TABLE** SQL statements.

You should consider SSDs if your I/O demands have outpaced the performance capabilities of traditional HDDs, latencies associated with spinning platters and moving arms limit the speed of HDD data access. SSDs near instantaneous data access removes this I/O bottleneck, creating a paradigm shift in I/O performance. Applications throttled by poor I/O performance can benefit greatly from SSDs.

Memory preference controls

Memory preference controls can be used as a technique to maximize performance and utilization of resources.

Memory preference controls

Memory preference controls can be used against performance critical database tables, indexes, physical files, and logical files as a technique to maximize performance and utilization of resources. Several approaches are available for controlling the memory preference:

1. **Set Object Access (SETOBJACC)** command

   One benefit of **SETOBJACC** is that you can carve out a separate memory pool that is not used by from any running applications or MEMORY_POOL_PREFERENCE and those objects will then not get paged out because neither applications nor SQE will be using that pool. If the target objects are primarily accessed using Native database I/O,**SETOBJACC** is the preferred approach.**SETOBJACC** uses a single thread to bring the object into memory.

2. **Change Physical File (CHGPF)** and **Change Logical File (CHGLF)** commands - Keep in memory (**KEEPINMEM**) parameter

   When an object is changed to have Keep in memory set to *YES, the database will bring the object into memory and attempt to keep it in memory when it is accessed using SQL via SQE. Native database I/O (for example RPG CHAIN, READ, etc.) does not do this.**KEEPINMEM** has the ability to use parallel I/O to bring the object into memory.

   - **CHGPF KEEPINMEM(*YES|*NO)**
   - **CHGLF KEEPINMEM(*YES|*NO)**

3. The SQL memory-preference can be used as an alternative to the **KEEPINMEM** command parameter.

   The behavior of SQL configured in memory objects matches the behavior described in the**KEEPINMEM** section.

   KEEP IN MEMORY <NO/YES> is available on the following SQL statements:

   - **ALTER TABLE**
   - **CREATE INDEX**
   - **CREATE TABLE**
   - **DECLARE GLOBAL TEMPORARY TABLE**

Note: The QSYS2/SYSPARTITIONSTAT and SYSPARTITIONINDEXSTAT catalogs can be queried to determine the memory-preference for specific objects. When a memory-preference is specified for an object, the MEMORY_POOL_PREFERENCE QAQQINI option can be used to influence where we attempt to page objects. There is no guarantee that objects will remain in memory.

Join optimization

A join operation is a complex function that requires special attention in order to achieve good performance. This section describes how Db2 for i implements join queries and how optimization choices are made by the query optimizer. It also describes design tips and techniques which help avoid or solve performance problems.

Nested loop join implementation

Db2 for i provides a **nested loop** join method. For this method, the processing of the tables in the join are ordered. This order is called the **join order**. The first table in the final join order is called the **primary table**. The other tables are called **secondary tables**. Each join table position is called a **dial**.

The nested loop is implemented either using an index on secondary tables, a hash table, or a table scan (arrival sequence) on the secondary tables. In general, the join is implemented using either an index or a hash table.

Index nested loop join implementation

During the join, Db2 for i:

1. Accesses the first primary table row selected by the predicates local to the primary table.
2. Builds a key value from the join columns in the primary table.
3. Chooses the access to the first secondary table:
   - If using an index, Radix Index Probe is used to locate the first row satisfying the join condition for the secondary table. The probe uses an index with keys matching the join condition or local row selection columns of the secondary table.
   - Applies bitmap selection, if applicable.

     All rows that satisfy the join condition from each secondary dial are located using an index. Rows are retrieved from secondary tables in random sequence. This random disk I/O time often accounts for a large percentage of the processing time of the query. Since a given secondary dial is searched once for each row selected from the primary and the preceding secondary dials that satisfy the join condition for each of the preceding secondary dials, many searches could be against the later dials. Any inefficiencies in the processing of the later dials can significantly inflate the query processing time. This reason is why attention to performance considerations for join queries can reduce the run time of a join query from hours to minutes.

     If an efficient index cannot be found, a temporary index could be created. Some join queries build temporary indexes over secondary dials even when an index exists for all the join keys. Because efficiency is important for secondary dials of longer running queries, the optimizer could build a temporary index containing only entries with local row selection for that dial. This preprocessing of row selection allows the database manager to process row selection in one pass instead of each time rows are matched for a dial.
   - If using a Hash Table Probe, a hash temporary result table is created containing all rows from local selection against the table on the first probe. The structure of the hash table is such that rows with the same join value are loaded into the same hash table partition (clustered). The location of the rows for any given join value can be found by applying a hashing function to the join value.

     A nested loop join using a Hash Table Probe has several advantages over a nested loop join using an Index Probe:

     – The structure of a hash temporary result table is simpler than the structure of an index. Less CPU processing is required to build and probe a hash table.

     – The rows in the hash result table contain all the data required by the query. There is no need to access the dataspace of the table with random I/O when probing the hash table.

     – Like join values are clustered, so all matching rows for a given join value can typically be accessed with a single I/O request.

     – The hash temporary result table can be built using SMP parallelism.

   - Unlike indexes, entries in hash tables are not updated to reflect changes of column values in the underlying table. The existence of a hash table does not affect the processing cost of other updating jobs in the system.
   - If using a Sorted List Probe, a sorted list result is created containing all the rows from local selection against the table on the first probe. The structure of the sorted list table is such that rows with the same join value are sorted together in the list. The location of the rows for any given join value can be found by probing using the join value.
   - If using a Table Scan, locate the first row that satisfies the join condition or local row selection columns of the secondary table. The join could be implemented with a table scan when the secondary table is a user-defined table function.
4. Determines if the row is selected by applying any remaining selection local to the first secondary dial.

   If the secondary dial row is not selected then the next row that satisfies the join condition is located. Steps 1 through 4 are repeated until a row that satisfies both the join condition and any remaining selection is selected from all secondary tables
5. Returns the result join row.
6. Processes the last secondary table again to find the next row that satisfies the join condition in that dial.

   During this processing, when no more rows satisfying the join condition can be selected, the processing backs up to the logical previous dial. It attempts to read the next row that satisfies its join condition.
7. Ends processing when all selected rows from the primary table are processed.

Note the following characteristics of a nested loop join:

- If ordering or grouping is specified, and all the columns are over a single table eligible to be the primary, then the optimizer costs the join with that table as the primary table, performing the grouping and ordering with an index.
- If ordering and grouping is specified on two or more tables or if temporary objects are allowed, Db2 for i breaks the processing of the query into two parts:
  1. Perform the join selection, omitting the ordering or grouping processing, and write the result rows to a temporary work table. This method allows the optimizer to consider any table of the join query as a candidate for the primary table.
  2. Perform the ordering or grouping on the data in the temporary work table.

Queries that cannot use hash join

Hash join cannot be used for queries that:

- Hash join cannot be used for queries involving physical files or tables that have read triggers.
- Require that the cursor position is restored as the result of the SQL ROLLBACK HOLD statement or the ROLLBACK CL command. For SQL applications using commitment control level other than *NONE, this method requires that *ALLREAD be specified as the value for the ALWBLK precompiler parameter.
- Hash join cannot be used for a table in a join query where the join condition something other than an equals operator.
- CQE does not support hash join if the query contains any of the following:
  - Subqueries unless all subqueries in the query can be transformed to inner joins.
  - UNION or UNION ALL
  - Perform left outer or exception join.
  - Use a DDS created join logical file.

Related concepts

Objects processed in parallel

The DB2 Symmetric multiprocessing feature provides the optimizer with additional methods for retrieving data that include parallel processing. Symmetrical multiprocessing is a form of parallelism achieved on a single system where multiple CPU and I/O processors sharing memory and disk work simultaneously toward a single result.

Related reference

Table scan
A table scan is the easiest and simplest operation that can be performed against a table. It sequentially processes all the rows in the table to determine if they satisfy the selection criteria specified in the query. It does this processing in a way to maximize the I/O throughput for the table.

Sorted list probe
A sorted list probe operation is used to retrieve rows from a temporary sorted list based upon a probe lookup operation.

Hash table probe
A hash table probe operation is used to retrieve rows from a temporary hash table based upon a probe lookup operation.

Radix index probe
A radix index probe operation is used to retrieve the rows from a table in a keyed sequence. The main difference between the radix index probe and the scan is that the rows returned are first identified by a probe operation to subset them.

Join optimization algorithm

The query optimizer must determine the join columns, join operators, local row selection, dial implementation, and dial ordering for a join query.

The join columns and join operators depend on the following situations:

- Join column specifications of the query
- Join order
- Interaction of join columns with other row selection

Join specifications not implemented for the dial are deferred until a later dial or, if an inner join, processed as row selection.

For a given dial, the only join specifications which are usable as join columns are those being joined to a *previous* dial. For example, the second dial can only use join specifications which reference columns in the primary dial. Likewise, the third dial can only use join specifications which reference columns in the primary and the second dials, and so on. Join specifications which reference later dials are deferred until the referenced dial is processed.

Note: For OPNQRYF, only one type of join operator is allowed for either a left outer or an exception join. That is, the join operator for all join conditions must be the same.

When looking for an existing index to access a secondary dial, the query optimizer looks at the left-most key columns of the index. For a given dial and index, the join specifications which use the left-most key columns can be used. For example:

```
DECLARE BROWSE2 CURSOR FOR
 SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.HIREDATE = EMP_ACT.EMSTDATE
 OPTIMIZE FOR 99999 ROWS
```

For the index over EMP_ACT with key columns EMPNO, PROJNO, and EMSTDATE, the join operation is performed only on column EMPNO. After the join is performed, index scan-key selection is done using column EMSTDATE.

The query optimizer also uses local row selection when choosing the best use of the index for the secondary dial. If the previous example had been expressed with a local predicate as:

```
DECLARE BROWSE2 CURSOR FOR
 SELECT * FROM EMPLOYEE, EMP_ACT
```

```
       WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
         AND EMPLOYEE.HIREDATE = EMP_ACT.EMSTDATE
         AND EMP_ACT.PROJNO = '123456'
      OPTIMIZE FOR 99999 ROWS
```

The index with key columns EMPNO, PROJNO, and EMSTDATE are fully utilized by combining join and selection into one operation against all three key columns.

When creating a temporary index, the left-most key columns are the usable join columns in that dial position. All local row selection for that dial is processed when selecting entries for inclusion into the temporary index. A temporary index is like the index created for a select/omit keyed logical file. The temporary index for the previous example has key columns of EMPNO and EMSTDATE.

Since the optimizer tries a combination of join and local row selection, you can achieve almost all the advantages of a temporary index by using an existing index. In the preceding example, using either implementation, an existing index could be used or a temporary index could be created. A temporary index is built with the local row selection on PROJNO applied during the index creation. The temporary index has key columns of EMPNO and EMSTDATE to match the join selection.

If, instead, an existing index was used with key columns of EMPNO, PROJNO, EMSTDATE (or PROJNO, EMP_ACT, EMSTDATE), the local row selection can be applied **at the same time** as the join selection. This method contrasts to applying the local selection before the join selection, as happens when the temporary index is created. Or applying the local selection after the join selection, as happens when only the first key column of the index matches the join column.

The existing index implementation is more likely to provide faster performance because join and selection processing are combined without the overhead of building a temporary index. However, the existing index could have slightly slower I/O processing than the temporary index because the local selection is run many times rather than once. In general, create indexes with key columns for the combination of join and equal selection columns as the left-most keys.

Join order optimization

The SQE optimizer allows join reordering for a join logical file. However, the join order is fixed if CQE runs a query that references a join logical file. The join order is also fixed if the OPNQRYF JORDER(*FILE) parameter is specified. In addition, the join order is fixed if the query options file (QAQQINI) FORCE_JOIN_ORDER parameter is *YES

Otherwise, the following join ordering algorithm is used to determine the order of the tables:

1. Determine an access method for each individual table as candidates for the primary dial.
2. Estimate the number of rows returned for each table based on local row selection.

   If the join query with ordering or grouping is processed in one step, the table with the ordering or grouping columns is the primary table.
3. Determine an access method, cost, and expected number of rows returned for each join combination of candidate tables as primary and first secondary tables.

   The join order combinations estimated for a four table inner join would be:

   ```
   1-2    2-1    1-3    3-1    1-4    4-1   2-3    3-2    2-4    4-2    3-4    4-3
   ```
4. Choose the combination with the lowest join cost and number of selected rows or both.
5. Determine the cost, access method, and expected number of rows for each remaining table joined to the previous secondary table.
6. Select an access method for each table that has the lowest cost for that table.
7. Choose the secondary table with the lowest join cost and number of selected rows or both.
8. Repeat steps 4 through 7 until the lowest cost join order is determined.

Note: After dial 32, the optimizer uses a different method to determine file join order, which might not be the lowest cost.

When a query contains a left or right outer join or a right exception join, the join order is not fixed. However, all from-columns of the ON clause must occur from dials previous to the left or right outer or exception join. For example:

```
FROM A INNER JOIN B ON A.C1=B.C1
LEFT OUTER JOIN C ON B. C2=C.C2
```

The allowable join order combinations for this query would be:

1–2–3, 2–1–3, or 2–3–1

Right outer or right exception joins are implemented as left outer and left exception, with files flipped. For example:

```
FROM A RIGHT OUTER JOIN B ON A.C1=B.C1
```

is implemented as B LEFT OUTER JOIN A ON B.C1=A.C1. The only allowed join order is 2–1.

Related information

Open Query File (OPNQRYF) command

Change Query Attributes (CHGQRYA) command

Full outer join

Full outer join is supported by the SQE optimizer. Just as right outer and right exception join are rewritten to the supported join types of inner, left outer or left exception, a full outer join is also rewritten.

A full outer join of A FULL OUTER JOIN B is equivalent to a (A LEFT OUTER JOIN B) UNION ALL (B LEFT EXCEPTION JOIN A). The following example illustrates the rewrite.

```
  SELECT EMPNO, LASTNAME, DEPTNAME
   FROM CORPDATA.EMPLOYEE XXX
   FULL OUTER JOIN CORPDATA.DEPARTMENT YYY
   ON XXX.WORKDEPT = YYY.DEPTNO
```

This query is rewritten as the following:

```
 SELECT EMPNO, LASTNAME, DEPTNAME
  FROM CORPDATA.EMPLOYEE XXX
  LEFT OUTER JOIN CORPDATA.DEPARTMENT YYY
  ON XXX.WORKDEPT = YYY.DEPTNO
 UNION ALL
 SELECT EMPNO, LASTNAME, DEPTNAME
  FROM CORPDATA.DEPARTMENT YYY
  LEFT EXCEPTION JOIN CORPDATA.EMPLOYEE XXX
  ON XXX.WORKDEPT = YYY.DEPTNO
```

A query with multiple FULL OUTER JOIN requests, such as `A FULL OUTER JOIN B FULL OUTER JOIN C` can quickly become complicated in this rewritten state. This complication is illustrated in the following example.

If not running in live data mode, the optimizer could facilitate performance both during optimization and runtime by encapsulating intermediate results in a temporary data object. This object can be optimized once and plugged into both the scanned and probed side of the rewrite. These shared temporary objects eliminate the need to make multiple passes through the specific tables to satisfy the request.

In this example, the result of the (A FULL OUTER JOIN B) is a candidate for encapsulation during its FULL OUTER join with C.

```
A FULL OUTER JOIN B  FULL OUTER JOIN C
```

This query is rewritten as the following:

```
 ((A LEFT OUTER JOIN B) UNION ALL (B LEFT EXCEPTION JOIN A)) LEFT OUTER JOIN C )
 UNION ALL
 (C LEFT EXCEPTION JOIN ((A LEFT OUTER JOIN B) UNION ALL (B LEFT EXCEPTION JOIN A))
```

FULL OUTER implies that both sides of the join request can generate NULL values in the resulting answer set. Local selection in the WHERE clause of the query could result in the appropriate downgrade of the FULL OUTER to a LEFT OUTER or INNER JOIN.

If you want FULL OUTER JOIN behavior and local selection applied, specify the local selection in the ON clause of the FULL OUTER JOIN, or use common table expressions. For example:

```
WITH TEMPEMP AS (SELECT * FROM CORPDATA.EMPLOYEE XXX WHERE SALARY > 10000)
SELECT EMPNO, LASTNAME, DEPTNAME
  FROM TEMPEMP XXX
  FULL OUTER JOIN CORPDATA.DEPARTMENT YYY
  ON XXX.WORKDEPT = YYY.DEPTNO
```

Join cost estimation and index selection

As the query optimizer compares the various possible access choices, it must assign a numeric cost value to each candidate. The optimizer uses that value to determine the implementation which consumes the least amount of processing time. This costing value is a combination of CPU and I/O time

In steps 3 and 5 in "Join order optimization" on page 62, the optimizer estimates cost and chooses an access method for a given dial combination. The choices made are like the choices for row selection, except that a plan using a probe must be chosen.

The costing value is based on the following assumptions:

- Table pages and index pages must be retrieved from auxiliary storage. For example, the query optimizer is not aware that an entire table might be loaded into active memory as the result of a **Set Object Access (SETOBJACC)** CL command. Use of this command could significantly improve the performance of a query. However, the optimizer does not change the query implementation to take advantage of the memory resident state of the table.
- The query is the only process running on the system. No allowance is given for system CPU utilization or I/O waits which occur because of other processes using the same resources. CPU-related costs are scaled to the relative processing speed of the system running the query.
- The values in a column are uniformly distributed across the table. For example, if 10% of the table rows have the same value, then on average, every 10th row in the table contains that value.
- The column values are independent from any other column values in a row, unless there is an index available whose key definition is (A, B). Multi-key field indexes allow the optimizer to detect when the values between columns are correlated.

  For example, a column named A has a value of 1 in 50% of the rows in a table. A column named B has a value of 2 in 50% of the rows. It is expected that a query which selects rows where `A = 1`, and `B = 2` selects 25% of the rows in the table.

The main factors in the join cost calculation for secondary dials are:

- the number of rows selected in all previous dials
- the number of rows which match, on average, each of the rows selected from previous dials.

Both of these factors can be derived by estimating the number of matching rows for a given dial.

When the join operator is something other than equal, the expected number of matching rows is based on the following default filter factors:

- 33% for less-than, greater-than, less-than-equal-to, or greater-than-equal-to
- 90% for not equal
- 25% for BETWEEN range (OPNQRYF %RANGE)
- 10% for each IN list value (OPNQRYF %VALUES)

For example, when the join operator is less-than, the expected number of matching rows is 0.33 * (number of rows in the dial). If no join specifications are active for the current dial, the Cartesian product is assumed to be the operator. For Cartesian products, the number of matching rows is every row in the dial, unless local row selection can be applied to the index.

When the join operator is equal, the expected number of rows is the average number of duplicate rows for a given value.

Related information

Set Object Access (SETOBJACC) command

Transitive closure predicates

For join queries, the query optimizer could do some special processing to generate additional selection. When the set of predicates that belong to a query logically infer extra predicates, the query optimizer generates additional predicates. The purpose is to provide more information during join optimization.

See the following examples:

```
SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.EMPNO = '000010'
```

The optimizer modifies the query to:

```
SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.EMPNO = '000010'
    AND EMP_ACT.EMPNO  = '000010'
```

The following rules determine which predicates are added to other join dials:

- The dials affected must have join operators of equal.
- The predicate is **isolatable**, which means that a false condition from this predicate omits the row.
- One operand of the predicate is an equal join column and the other is a constant or host variable.
- The predicate operator is not LIKE (OPNQRYF %WLDCRD, or *CT).
- The predicate is not connected to other predicates by OR.

The query optimizer generates a new predicate, whether a predicate exists in the WHERE clause (OPNQRYF QRYSLT parameter).

Some predicates are redundant. Redundant predicates occur when a previous evaluation of other predicates in the query already determines the result that predicate provides. Redundant predicates can be specified by you or generated by the query optimizer during predicate manipulation. Redundant predicates with operators of =, >, >=, <, <=, or BETWEEN (OPNQRYF *EQ, *GT, *GE, *LT, *LE, or %RANGE) are merged into a single predicate to reflect the most selective range.

Look ahead predicate generation (LPG)

A special type of transitive closure called look ahead predicate generation (LPG) might be costed for joins. In this case, the optimizer tries to minimize the random I/O of a join by pre-applying the query results to a large fact table. LPG is typically used with a class of queries referred to as star join queries. However, it can possibly be used with any join query.

Look at the following query:

```
SELECT * FROM EMPLOYEE,EMP_ACT
    WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.EMPNO ='000010'
```

The optimizer could decide to internally modify the query to be:

```
WITH HT AS (SELECT *
        FROM EMPLOYEE
        WHERE EMPLOYEE.EMPNO='000010')

SELECT *
    FROM HT, EMP_ACT
    WHERE HT.EMPNO = EMP_ACT.EMPNO
    AND EMP_ACT.EMPNO IN (SELECT DISTINCT EMPNO
                FROM HT)
```

The optimizer places the results of the "subquery" into a temporary hash table. The hash table of the subquery can be applied in one of two methods against the EMP_ACT (fact) table:

- The distinct values of the hash tables are retrieved. For each distinct value, an index over EMP_ACT is probed to determine which records are returned for that value. Those record identifiers are normally then stored and sorted (sometimes the sorting is omitted, depending on the total number of record ids expected). Once the ids are determined, the subset of EMP_ACT records can be accessed more efficiently than in a traditional nested loop join processing.
- EMP_ACT can be scanned. For each record, the hash table is probed to see if the record joins at all to EMPLOYEE. This method allows for efficient access to EMP_ACT with a more efficient record rejection method than in a traditional nested loop join process.

Note: LPG processing is part of the normal processing in the SQL Query Engine. CQE only considers the first method, requires that the index in question by an EVI and also requires use of the STAR_JOIN and FORCE_JOIN_ORDER QAQQINI options.

Tips for improving performance when selecting data from more than two tables

The following suggestion is only applicable to CQE and is directed specifically to select-statements that access several tables. For joins that involve more than two tables, you might want to provide redundant information about the join columns. The CQE optimizer does not generate transitive closure predicates between two columns. If you give the optimizer extra information to work with when requesting a join, it can determine the best way to do the join. The additional information might seem redundant, but is helpful to the optimizer.

If the select-statement you are considering accesses two or more tables, all the recommendations suggested in "Creating an index strategy" on page 204 apply. For example, instead of coding:

```
EXEC SQL
  DECLARE EMPACTDATA CURSOR FOR
  SELECT LASTNAME, DEPTNAME, PROJNO, ACTNO
       FROM CORPDATA.DEPARTMENT, CORPDATA.EMPLOYEE,
              CORPDATA.EMP_ACT
      WHERE DEPARTMENT.MGRNO = EMPLOYEE.EMPNO
             AND EMPLOYEE.EMPNO = EMP_ACT.EMPNO
END-EXEC.
```

Provide the optimizer with a little more data and code:

```
EXEC SQL
  DECLARE EMPACTDATA CURSOR FOR
  SELECT LASTNAME, DEPTNAME, PROJNO, ACTNO
       FROM CORPDATA.DEPARTMENT, CORPDATA.EMPLOYEE,
              CORPDATA.EMP_ACT
       WHERE DEPARTMENT.MGRNO = EMPLOYEE.EMPNO
              AND EMPLOYEE.EMPNO = EMP_ACT.EMPNO
              AND DEPARTMENT.MGRNO = EMP_ACT.EMPNO
END-EXEC.
```

Multiple join types for a query

Multiple join types (inner, left outer, right outer, left exception, and right exception) can be specified in the query using the JOIN syntax. However, the Db2 for i can only support one join type of inner, left outer, or left exception join for the entire query. The optimizer determines the overall join type for the query and reorders the files to achieve the correct semantics.

Note: This section does not apply to SQE or OPNQRYF.

The optimizer evaluates the join criteria, along with any row selection, to determine the join type for each dial and the entire query. Then the optimizer generates additional selection using the relative row number of the tables to simulate the different types of joins that occur within the query.

Null values are returned for any unmatched rows in either a left outer or an exception join. Any isolatable selection specified for that dial, including any additional join criteria specified in the WHERE clause, causes all the unmatched rows to be eliminated. (The exception is when the selection is for an IS NULL predicate.) This elimination causes the dial join type to change to an inner join (or an exception join) if the IS NULL predicate was specified.

In the following example, a left outer join is specified between the tables EMPLOYEE and DEPARTMENT. In the WHERE clause, there are two selection predicates that also apply to the DEPARTMENT table.

```
     SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
      FROM CORPDATA.EMPLOYEE XXX LEFT OUTER JOIN CORPDATA.DEPARTMENT YYY
              ON XXX.WORKDEPT = YYY.DEPTNO
            LEFT OUTER JOIN CORPDATA.PROJECT ZZZ
              ON XXX.EMPNO = ZZZ.RESPEMP
      WHERE XXX.EMPNO = YYY.MGRNO AND
              YYY.DEPTNO IN ('A00', 'D01', 'D11', 'D21', 'E11')
```

The first selection predicate, XXX.EMPNO = YYY.MGRNO, is an additional join condition that is evaluated as an "inner join" condition. The second is an isolatable selection predicate that eliminates any unmatched rows. Either of these predicates can cause the join type for the DEPARTMENT table to change from a left outer join to an inner join.

Even though the join between the EMPLOYEE and DEPARTMENT tables was changed to an inner join, the entire query remains a left outer join to satisfy the join condition for the PROJECT table.

Note: Care must be taken when specifying multiple join types since they are supported by appending selection to the query for any unmatched rows. The number of rows satisfying the join criteria can become large before selection that either selects or omits the unmatched rows based on that individual dial join type is applied.

Sources of join query performance problems
The optimization algorithms described earlier benefit most join queries, but the performance of a few queries might be degraded.

This occurs when:

- An index is not available which provides average number of duplicate values statistics for the potential join columns.
- The optimizer uses default filter factors to estimate the number of rows when applying local selection to the table when indexes or column statistics do not exist over the selection columns.

  Creating indexes over the selection columns allows the optimizer to make a more accurate filtering estimate by using key range estimates.
- The particular values selected for the join columns yield a greater number of matching rows than the average number of duplicate values for all values of the join columns in the table. For example, the data is not uniformly distributed.

Join performance tips

If you have a join query performing poorly, or you are creating an application which uses join queries, these tips could be useful.

Table 27. Checklist for Creating an Application that Uses Join Queries

What to Do	How It Helps
Check the database design. Make sure that there are indexes available over all the join columns and row selection columns or both. The optimizer provides index advice in several places to aid in this process: • the index advisor under System i Navigator - Database • the advised information under Visual Explain • the advised information in the 3020 record in the database monitor	The query optimizer can select an efficient access method because it can determine the average number of duplicate values. Many queries could use the existing index and avoid the cost of creating a temporary index or hash table.
Check the query to see whether some complex predicates could be added to other dials to allow the optimizer to get better selectivity for each dial.	The query optimizer does not add predicates for predicates connected by OR or non-isolatable predicates, or predicate operator LIKE. Modify the query by adding additional predicates to help.
Specify ALWCPYDTA(*OPTIMIZE) or ALWCPYDTA(*YES)	The query is creating a temporary index or hash table, and the processing time could be better if the existing index or hash table was used. Specify ALWCPYDTA(*YES). The query is not creating a temporary index or hash table, and the processing time could be better if a temporary index was created. Specify ALWCPYDTA(*OPTIMIZE). Alternatively, specify OPTIMIZE FOR n ROWS to inform the optimizer that the application reads every resulting row. Set n to a large number. You can also set n to a small number before ending the query.
For OPNQRYF, specify OPTIMIZE(*FIRSTIO) or OPTIMIZE(*ALLIO)	Specify the OPTIMIZE(*FIRSTIO) or OPTIMIZE(*ALLIO) option to accurately reflect your application. Use *FIRSTIO, if you want the optimizer to optimize the query to retrieve the first block of rows most efficiently. This biases the optimizer toward using existing objects. If you want to optimize the retrieval time for the entire answer set, use *ALLIO. This option could cause the optimizer to create temporary indexes or hash tables to minimize I/O.

Table 27. Checklist for Creating an Application that Uses Join Queries (continued)	
What to Do	**How It Helps**

Star join queries	A join in which one table is joined with all secondary tables consecutively is sometimes called a **star join**. If all secondary join predicates contain a column reference to a particular table, place that table in join position one. In Example A, all tables are joined to table EMPLOYEE. The query optimizer can freely determine the join order. For SQE, the optimizer uses Look Ahead Predicate generation to determine the optimal join order. For CQE, the query could be changed to force EMPLOYEE into join position one by using the query options file (QAQQINI) FORCE_JOIN_ORDER parameter of *YES. In these examples, the join type is a join with no default values returned (an inner join.). The reason for forcing the table into the first position is to avoid random I/O processing. If EMPLOYEE is not in join position one, every row in EMPLOYEE can be examined repeatedly during the join process. If EMPLOYEE is fairly large, considerable random I/O processing occurs resulting in poor performance. By forcing EMPLOYEE to the first position, random I/O processing is minimized.
	
	Example A: Star join query
	
	`DECLARE C1 CURSOR FOR`
	`  SELECT * FROM DEPARTMENT, EMP_ACT, EMPLOYEE,`
	`  PROJECT`
	`  WHERE DEPARTMENT.DEPTNO=EMPLOYEE.WORKDEPT`
	`  AND EMP_ACT.EMPNO=EMPLOYEE.EMPNO`
	`  AND EMPLOYEE.WORKDEPT=PROJECT.DEPTNO`
	
	Example B: Star join query with order forced using FORCE_JOIN_ORDER
	
	`  DECLARE C1 CURSOR FOR`
	`  SELECT * FROM EMPLOYEE, DEPARTMENT, EMP_ACT,`
	`  PROJECT`
	`  WHERE DEPARTMENT.DEPTNO=EMPLOYEE.WORKDEPT`
	`  AND EMP_ACT.EMPNO=EMPLOYEE.EMPNO`
	`  AND EMPLOYEE.WORKDEPT=PROJECT.DEPTNO`

Specify ALWCPYDTA(*OPTIMIZE) to allow the query optimizer to use a sort routine.	Ordering is specified and all key columns are from a single dial. The optimizer can consider all possible join orders with ALWCPYDTA(*OPTIMIZE).
Specify join predicates to prevent all the rows from one table from being joined to every row in the other table.	Improves performance by reducing the join fan-out. It is best if every secondary table has at least one join predicate that references one of its columns as a 'join-to' column.

Distinct optimization

Distinct is used to compare a value with another value.

There are two methods to write a query that returns distinct values in SQL. One method uses the DISTINCT keyword:

```
SELECT DISTINCT COL1, COL2
    FROM TABLE1
```

The second method uses GROUP BY:

```
SELECT COL1, COL2
    FROM TABLE1
    GROUP BY COL1, COL2
```

All queries that contain a DISTINCT, and are run using SQE, rewritten into queries using GROUP BY. This rewrite enables queries using DISTINCT to take advantage of the many grouping techniques available to the optimizer.

Distinct to Grouping implementation

The following example query has a DISTINCT:

```
SELECT DISTINCT COL1, COL2
    FROM T1
    WHERE COL2 > 5 AND COL3 = 2
```

The optimizer rewrites it into this query:

```
SELECT COL1, COL2
    FROM T1
    WHERE COL2 > 5 AND COL3 = 2
    GROUP BY COL1, COL2
```

Distinct removal

A query containing a DISTINCT over whole-file aggregation (no grouping or selection) allows the DISTINCT to be removed. For example, look at this query with DISTINCT:

```
SELECT DISTINCT COUNT(C1), SUM(C1)
    FROM TABLE1
```

The optimizer rewrites this query as the following:

```
SELECT COUNT(C1), SUM(C1)
    FROM TABLE1
```

If the DISTINCT and the GROUP BY fields are identical, the DISTINCT can be removed. If the DISTINCT fields are a subset of the GROUP BY fields (and there are no aggregates), the DISTINCTs can be removed.

Grouping optimization

Db2 for i has certain techniques to use when the optimizer encounters grouping. The query optimizer chooses its methods for optimizing your query.

Hash grouping implementation

This technique uses the base hash access method to perform grouping or summarization of the selected table rows. For each selected row, the specified grouping value is run through the hash function. The computed hash value and grouping value are used to quickly find the entry in the hash table corresponding to the grouping value.

If the current grouping value already has a row in the hash table, the hash table entry is retrieved and summarized (updated) with the current table row values based on the requested grouping column operations (such as SUM or COUNT). If a hash table entry is not found for the current grouping value, a new entry is inserted into the hash table and initialized with the current grouping value.

The time required to receive the first group result for this implementation is most likely longer than other grouping implementations because the hash table must be built and populated first. Once the hash table is populated, the database manager uses the table to start returning the grouping results. Before returning any results, the database manager must apply any specified grouping selection criteria or ordering to the summary entries in the hash table.

Where the hash grouping method is most effective

The hash grouping method is most effective when the consolidation ratio is high. The **consolidation ratio** is the ratio of the selected table rows to the computed grouping results. If every database table row has its own unique grouping value, then the hash table becomes too large. The size in turn slows down the hashing access method.

The optimizer estimates the consolidation ratio by first determining the number of unique values in the specified grouping columns (that is, the expected number of groups in the database table). The optimizer then examines the total number of rows in the table and the specified selection criteria and uses the result of this examination to estimate the consolidation ratio.

Indexes over the grouping columns can help make the ratio estimate of the optimizer more accurate. Indexes improve the accuracy because they contain statistics that include the average number of duplicate values for the key columns.

The optimizer also uses the expected number of groups estimate to compute the number of partitions in the hash table. As mentioned earlier, the hashing access method is more effective when the hash table is well-balanced. The number of hash table partitions directly affects how entries are distributed across the hash table and the uniformity of this distribution.

The hash function performs better when the grouping values consist of columns that have non-numeric data types, except for the integer (binary) data type. In addition, specifying grouping value columns that are not associated with the variable length and null column attributes allows the hash function to perform more effectively.

Index grouping implementation
There are two primary ways to implement grouping using an index: Ordered grouping and pre-summarized processing.

Ordered grouping

This implementation uses the Radix Index Scan or the Radix Index Probe access methods to perform the grouping. An index is required that contains all the grouping columns as contiguous leftmost key columns. The database manager accesses the individual groups through the index and performs the requested summary functions.

Since the index, by definition, already has all the key values grouped, the first group result can be returned in less time than the hashing method. This index performance is faster because the hashing method requires a temporary result. This implementation can be beneficial if an application does not need to retrieve all the group results, or if an index exists that matches the grouping columns.

When the grouping is implemented with an index and a permanent index does not exist that satisfies grouping columns, a temporary index is created. The grouping columns specified within the query are used as the key columns for this index.

Pre-summarized processing

This SQE-only implementation uses an Encoded Vector Index to extract the summary information already in the symbol table of the index. The EVI symbol table contains the unique key values and a count of the number of table records that have that unique value. The grouping for the columns of the index key is already performed. If the query references a single table and performs simple aggregation, the EVI might be used for quick access to the grouping results. For example, consider the following query:

```
SELECT COUNT(*), col1
    FROM t1
    GROUP BY col1
```

If an EVI exists over t1 with a key of col1, the optimizer can rewrite the query to access the precomputed grouping answer in the EVI symbol table.

This rewrite can result in dramatic improvements when the number of table records is large and the number of resulting groups is small, relative to the size of the table.

This method is also possible with selection (WHERE clause), as long as the reference columns are in the key definition of the EVI.

For example, consider the following query:

```
SELECT COUNT(*), col1
 FROM t1
```

```
 WHERE col1 > 100
 GROUP BY col1
```

This query can be rewritten by the optimizer to use the EVI. This pre-summarized processing works for DISTINCT processing, GROUP BY and for column function COUNT. All columns of the table referenced in the query must also be in the key definition of the EVI.

So, for example, the following query can be made to use the EVI:

```
SELECT DISTINCT col1
    FROM t1
```

However, this query cannot:

```
SELECT DISTINCT col1
 FROM t1
 WHERE col2 > 1
```

This query cannot use the EVI because it references col2 of the table, which is not in the key definition of the EVI. If multiple columns are defined in the EVI key, for example, col1 and col2, it is important to use the left-most columns of the key. For example, if an EVI existed with a key definition of (col1, col2), but the query referenced only col2, it is unlikely the EVI is used.

EVI INCLUDE aggregates

A more powerful example of pre-summarized processing can be facilitated by the use of the INCLUDE keyword on the index create. By default, COUNT(*) is implied on the creation of an EVI. Additional numeric aggregates specified over non-key data can further facilitate pre-determined or ready-made aggregate results during query optimization.

For example, suppose the following query is a frequently requested result set, queried in whole or as part of a subquery comparison.

```
SELECT  AVG(col2)
FROM t1
GROUP BY col1
```

Create the following EVI to predetermine the value of AVG(col2).

```
CREATE ENCODED VECTOR INDEX  eviT1 ON t1(col1) INCLUDE(AVG(col2))
```

eviT1 delivers distinct values for col1 and COUNT(*) specific to the group by of col1. eviT1 can be used to generate an asynchronous bitmap or RRN list for accessing the table rows for specific col1 values. In addition, eviT1 computes an additional aggregate, AVG(col2), over the same group by column (col1) by specifying the INCLUDE aggregate.

INCLUDE aggregates are limited to those aggregates that result in numeric values: SUM, COUNT, AVG, STDDEV, and so on. These values can be readily maintained as records are inserted, deleted, or updated in the base table.

MIN or MAX are two aggregates that are not supported as INCLUDE aggregates. Deleting the current row contributing to the MIN or MAX value would result in the need to recalculate, potentially accessing many rows, and reducing performance.

INCLUDE values can also contain aggregates over derivations. For example, if you have a couple of columns that contribute to an aggregate, that derivation can be specified, for example, as SUM(col1+col2+col3).

It is recommended that EVIs with INCLUDE aggregates only contain references to columns or column-specific derivations, for example, SUM(salary+bonus).

In many environments, queries that contain derivations using constants convert those constants to parameter markers. This conversion allows a much higher degree of ODP reuse. However, it can be more difficult to match the parameter value to a literal in the index definition.

The optimizer does attempt to match constants in the EVI with parameter markers or host variable values in the query. However, in some complex cases this support is limited and could result in the EVI not matching the query.

Pre-summarized processing can also take advantage of EVIs with INCLUDE in a JOIN situation.

For example, see the following aggregate query over the join of two tables.

EVI INCLUDE aggregate example

```
SELECT deptname, sum(salary)
FROM DEPARTMENT, EMPLOYEE
WHERE deptno=workdept
GROUP BY deptname
```

By providing an EVI with INCLUDE index, as follows, and with optimizer support to push down aggregates to the table level when possible, the resulting implementation takes advantage of the ready-made aggregates already supplied by EVI employeeSumByDept. The implementation never needs to touch or aggregate rows in the Employee table.

```
CREATE ENCODED VECTOR INDEX employeeSumByDept ON employee(workdept)
INCLUDE(sum(salary))
```

Aggregate pushdown results in a rewrite with EVI INCLUDE implementation, conceptually like the following query.

```
SELECT deptname, sum(sum(salary))
FROM department,
     (SELECT  workdept, sum(salary) FROM employee group by workdept) employee_2
WHERE deptno=workdept
```

Instead of department joining to all the rows in the employee table, it now has the opportunity to join to the predetermined aggregates, the sum of salary by department number, in the EVI symbol table. This results in significant reduction in processing and IO.

Related concepts

How the EVI works
EVIs work in different ways for costing and implementation.

Related reference

Encoded vector index symbol table scan
An encoded vector index symbol table scan operation is used to retrieve the entries from the symbol table portion of the index.

Related information

SQL INCLUDE statement

Optimizing grouping by eliminating grouping columns

All the grouping columns are evaluated to determine if they can be removed from the list of grouping columns. Only those grouping columns that have isolatable selection predicates with an equal operator specified can be considered. This guarantees that the column can only match a single value and does not help determine a unique group.

This processing allows the optimizer to consider more indexes to implement the query. It also reduces the number of columns that are added as key columns to a temporary index or hash table.

The following example illustrates a query where the optimizer might eliminate a grouping column.

```
     DECLARE DEPTEMP CURSOR FOR
       SELECT EMPNO, LASTNAME, WORKDEPT
         FROM CORPDATA.EMPLOYEE
         WHERE EMPNO = '000190'
         GROUP BY EMPNO, LASTNAME, WORKDEPT
```

OPNQRYF example:

```
OPNQRYF FILE(EMPLOYEE) FORMAT(FORMAT1)
   QRYSLT('EMPNO *EQ ''000190''')
   GRPFLD(EMPNO LASTNAME WORKDEPT)
```

In this example, the optimizer can remove EMPNO from the list of grouping columns because of the `EMPNO = '000190'` selection predicate. An index that only has LASTNAME and WORKDEPT specified as key columns could implement the query. If a temporary index or hash is required then EMPNO is not used.

Note: Even though EMPNO can be removed from the list of grouping columns, the optimizer might use a permanent index that exists with all three grouping columns.

Optimizing grouping by adding additional grouping columns
The same logic that is applied to removing grouping columns can also be used to add additional grouping columns to the query. Additional grouping columns are added only when you are trying to determine if an index can be used to implement the grouping.

The following example illustrates a query where the optimizer might add an additional grouping column.

```
    CREATE INDEX X1 ON EMPLOYEE
          (LASTNAME, EMPNO, WORKDEPT)

    DECLARE DEPTEMP CURSOR FOR
      SELECT LASTNAME, WORKDEPT
        FROM CORPDATA.EMPLOYEE
        WHERE EMPNO = '000190'
        GROUP BY LASTNAME, WORKDEPT
```

For this query request, the optimizer can add EMPNO as an additional grouping column when considering X1 for the query.

Optimizing grouping by using index skip key processing
Index Skip Key processing can be used when grouping with the keyed sequence implementation algorithm which uses an existing index. It is a specialized version of ordered grouping that processes few records in each group rather than all records in each group.

The index skip key processing algorithm:

1. Uses the index to position to a group and
2. finds the first row matching the selection criteria for the group, and if specified the first non-null MIN or MAX value in the group
3. Returns the group to the user
4. "Skip" to the next group and repeat processing

This algorithm improves performance by potentially not processing all index key values for a group.

Index skip key processing can be used:

- For single table queries using the keyed sequence grouping implementation when:
  - There are no column functions in the query, or
  - There is only a single MIN or MAX column function and the MIN or MAX operand is the next index key column after the grouping columns. There can be no other grouping functions in the query. For the MIN function, the key column must be an ascending key; for the MAX function, the key column must be a descending key. If the query is whole table grouping, the operand of the MIN or MAX must be the first key column.

    Example 1, using SQL:

    ```
    CREATE INDEX IX1 ON EMPLOYEE (SALARY DESC)
    ```

```
DECLARE C1 CURSOR FOR
   SELECT MAX(SALARY) FROM EMPLOYEE;
```

The query optimizer chooses to use the index IX1. The SLIC runtime code scans the index until it finds the first non-null value for SALARY. Assuming that SALARY is not null, the runtime code positions to the first index key and return that key value as the MAX of salary. No more index keys are processed.

Example 2, using SQL:

```
CREATE INDEX IX2 ON EMPLOYEE (WORKDEPT, JOB, SALARY)

DECLARE C1 CURSOR FOR
    SELECT WORKDEPT, MIN(SALARY)
        FROM EMPLOYEE
        WHERE JOB='CLERK'
        GROUP BY WORKDEPT
```

The query optimizer chooses to use Index IX2. The database manager positions to the first group for DEPT where JOB equals 'CLERK' and returns the SALARY. The code then skips to the next DEPT group where JOB equals 'CLERK'.

- For join queries:
  - All grouping columns must be from a single table.
  - For each dial, there can be at most one MIN or MAX column function operand that references the dial. No other column functions can exist in the query.
  - If the MIN or MAX function operand is from the same dial as the grouping columns, then it uses the same rules as single table queries.
  - If the MIN or MAX function operand is from a different dial, then the join column for that dial must join to one of the grouping columns. The index for that dial must contain the join columns followed by the MIN or MAX operand.

  Example 1, using SQL:

```
CREATE INDEX IX1 ON DEPARTMENT(DEPTNAME)

CREATE INDEX IX2 ON EMPLOYEE(WORKDEPT, SALARY)

DECLARE C1 CURSOR FOR
   SELECT DEPARTMENT.DEPTNO, MIN(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DEPARTMENT.DEPTNO=EMPLOYEE.WORKDEPT
      GROUP BY DEPARTMENT.DEPTNO;
```

Optimizing grouping by removing read triggers

For queries involving physical files or tables with read triggers, group by triggers always involve a temporary file before the group by processing. Therefore, these queries slow down.

Note: Read triggers are added when the **Add Physical File Trigger (ADDPFTRG)** command has been used on the table with TRGTIME (*AFTER) and TRGEVENT (*READ).

The query runs faster if the read trigger is removed (RMVPFTRG TRGTIME (*AFTER) TRGEVENT (*READ)).

Related information

Add Physical File Trigger (ADDPFTRG) command

Grouping set optimization

The optimizer uses all the previously mentioned grouping optimizations for individual grouping sets specified in the query.

If multiple temporary result sets are needed to implement all the grouping sets, they can all be populated using one pass through the data. This one-pass population occurs even if different types of temporary result sets are used to implement various grouping sets.

A temporary result type called sorted distinct list is used specifically for ROLLUP implementations. This temporary result set is used to compute the aggregate rows: the grouping set that includes all expressions listed in the ROLLUP clause. Hash grouping is used internally to quickly find the current grouping value. The entries in the temporary result sets are also sorted. This sorting allows the aggregate results to be used to compute the super-aggregate rows in the rollup result set without creating additional temporary result sets.

ROLLUPs can also be implemented using a radix index over the columns in the rollup without creating a temporary result set.

The optimizer can compute all the grouping sets in a given ROLLUP using at most one temporary result set. Therefore, it is advantageous for the optimizer to look for the rollup pattern in grouping set queries.

The optimizer tries to find the ROLLUP pattern in a list of individual grouping sets. For example, the following GROUP BY clause:

```
GROUP BY GROUPING SETS ((A, B, C), (B, D), (A, B), (A), ())
```

is rewritten to:

```
GROUP BY GROUPING SETS ((ROLLUP(A, B, C)), (B, D))
```

This rewrite allows the query to be implemented using at most two temporary results sets rather than 4.

Queries containing a CUBE clause is broken down into a union of ROLLUPs and grouping sets. For example:

```
CUBE(A, B, C)
```

is equivalent to:

```
(ROLLUP(A, B, C)),  (ROLLUP'(B, C)),  (ROLLUP'(C, A))
```

The ROLLUP' notation is an internal representation of a ROLLUP operation that does not include a grand total row in its result set. So, ROLLUP'(B, C) is equivalent to GROUP BY GROUPING SETS ((B,C), (B)). This CUBE rewrite implements at most three temporary result sets, rather than the 8 that might be needed had the query not been rewritten.

Ordering optimization

This section describes how Db2 for i implements ordering techniques, and how optimization choices are made by the query optimizer. The query optimizer can use either index ordering or a sort to implement ordering.

Sort Ordering implementation

The sort algorithm reads the rows into a sort space and sorts the rows based on the specified ordering keys. The rows are then returned to the user from the ordered sort space.

Index Ordering implementation

The index ordering implementation requires an index that contains all the ordering columns as contiguous leftmost key columns. The database manager accesses the individual rows through the index in index order, which results in the rows being returned in order to the requester.

This implementation can be beneficial if an application does not need to retrieve all the ordered results, or if an index exists that matches the ordering columns. When the ordering is implemented with an index, and a permanent index does not exist that satisfies ordering columns, a temporary index is created. The ordering columns specified within the query are used as the key columns for this index.

Optimizing ordering by eliminating ordering columns

All the ordering columns are evaluated to determine if they can be removed from the list of ordering columns. Only those ordering columns that have isolatable selection predicates with an equal operator specified can be considered. This guarantees that the column can match only a single value, and does not help determine in the order.

The optimizer can now consider more indexes as it implements the query. The number of columns that are added as key columns to a temporary index is also reduced. The following SQL example illustrates a query where the optimizer might eliminate an ordering column.

```
DECLARE DEPTEMP CURSOR FOR
  SELECT EMPNO, LASTNAME, WORKDEPT
  FROM CORPDATA.EMPLOYEE
  WHERE EMPNO = '000190'
  ORDER BY EMPNO, LASTNAME, WORKDEPT
```

Optimizing ordering by adding additional ordering columns

The same logic that is applied to removing ordering columns can also be used to add additional grouping columns to the query. This logic is done only when you are trying to determine if an index can be used to implement the ordering.

The following example illustrates a query where the optimizer might add an additional ordering column.

```
CREATE INDEX X1 ON EMPLOYEE (LASTNAME, EMPNO, WORKDEPT)

DECLARE DEPTEMP CURSOR FOR
  SELECT LASTNAME, WORKDEPT
  FROM CORPDATA.EMPLOYEE
  WHERE EMPNO = '000190'
  ORDER BY LASTNAME, WORKDEPT
```

For this query request, the optimizer can add EMPNO as an additional ordering column when considering X1 for the query.

View implementation

Views, derived tables (nested table expressions or NTEs), and common table expressions (CTEs) are implemented by the query optimizer using one of two methods.

These methods are:

- The optimizer combines the query select statement with the select statement of the view.
- The optimizer places the results of the view in a temporary table and then replaces the view reference in the query with the temporary table.

View composite implementation

The view composite implementation takes the query select statement and combines it with the select statement of the view to generate a new query. The new, combined select statement query is then run directly against the underlying base tables.

This single, composite statement is the preferred implementation for queries containing views, since it requires only a single pass of the data.

See the following examples:

```
CREATE VIEW  D21EMPL AS
  SELECT * FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT='D21'
```

Using SQL:

```
  SELECT LASTNAME, FIRSTNME, SALARY
  FROM D21EMPL
  WHERE JOB='CLERK'
```

The query optimizer generates a new query that looks like the following example:

```
SELECT LASTNAME, FIRSTNME, SALARY
  FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT='D21' AND JOB='CLERK'
```

The query contains the columns selected by the user query, the base tables referenced in the query, and the selection from both the view and the user query.

Note: The new composite query that the query optimizer generates is not visible to users. Only the original query against the view is seen by users and database performance tools.

View materialization implementation

The view materialization implementation runs the query of the view and places the results in a temporary result. The view reference in the user query is then replaced with the temporary, and the query is run against the temporary result.

View materialization is done whenever it is not possible to create a view composite. For SQE, view materialization is optional. The following types of queries require view materialization:

- The outermost view select contains grouping, the query contains grouping, and refers to a column derived from a column function in the view HAVING or select-list.
- The query is a join and the outermost select of the view contains grouping or DISTINCT.
- The outermost select of the view contains DISTINCT, and the query has UNION, grouping, or DISTINCT and one of the following:
  - Only the query has a shared weight NLSS table
  - Only the view has a shared weight NLSS table
  - Both the query and the view have a shared weight NLSS table, but the tables are different.
- The query contains a column function and the outermost select of the view contains a DISTINCT
- The view does not contain an access plan. Occurs when a view references a view, and a view composite cannot be created because of one of the previous listed reasons. Does not apply to nested table expressions and common table expressions.
- The Common table expression (CTE) is referenced more than once in the query FROM clause. Also, the CTE SELECT clause references a MODIFIES or EXTERNAL ACTION UDF.

When a temporary result table is created, access methods that are allowed with ALWCPYDTA(*OPTIMIZE) could be used to implement the query. These methods include hash grouping, hash join, and bitmaps.

See the following examples:

```
CREATE VIEW AVGSALVW AS
  SELECT WORKDEPT, AVG(SALARY) AS AVGSAL
  FROM CORPDATA.EMPLOYEE
  GROUP BY WORKDEPT
```

SQL example:

```
  SELECT D.DEPTNAME, A.AVGSAL
  FROM CORPDATA.DEPARTMENT D, AVGSALVW A
  WHERE D.DEPTNO=A.WORKDEPT
```

In this case, a view composite cannot be created since a join query references a grouping view. The results of AVGSALVW are placed in a temporary result table (*QUERY0001). The view reference AVGSALVW is replaced with the temporary result table. The new query is then run. The generated query looks like the following:

```
SELECT D.DEPTNAME, A.AVGSAL
  FROM CORPDATA.DEPARTMENT D, *QUERY0001 A
  WHERE D.DEPTNO=A.WORKDEPT
```

Note: The new query that the query optimizer generates is not visible to users. Only the original query against the view is seen by users and database performance tools.

Whenever possible, isolatable selection from the query, except subquery predicates, is added to the view materialization process. This results in smaller temporary result tables and allows existing indexes to be used when materializing the view. This process is not done if there is more than one reference to the same view or common table expression in the query. The following is an example where isolatable selection is added to the view materialization:

```
SELECT D.DEPTNAME,A.AVGSAL
  FROM CORPDATA.DEPARTMENT D, AVGSALVW A
  WHERE D.DEPTNO=A.WORKDEPT AND
  A.WORKDEPT LIKE 'D%' AND AVGSAL>10000
```

The isolatable selection from the query is added to the view resulting in a new query to generate the temporary result table:

```
SELECT WORKDEPT, AVG(SALARY) AS AVGSAL
  FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT LIKE 'D%'
  GROUP BY WORKDEPT
  HAVING AVG(SALARY)>10000
```

Materialized query table optimization

Materialized query tables (MQTs) (also referred to as automatic summary tables or materialized views) can provide performance enhancements for queries.

This performance enhancement is done by precomputing and storing results of a query in the materialized query table. The database engine can use these results instead of recomputing them for a user specified query. The query optimizer looks for any applicable MQTs. The optimizer can implement the query using a given MQT, provided it is a faster implementation choice.

Materialized Query Tables are created using the SQL CREATE TABLE statement. Alternatively, the ALTER TABLE statement could be used to convert an existing table into a materialized query table. The REFRESH TABLE statement is used to recompute the results stored in the MQT. For user-maintained MQTs, the MQTs could also be maintained by the user using INSERT, UPDATE, and DELETE statements.

Related information

Create Table statement

MQT supported function

Although an MQT can contain almost any query, the optimizer only supports a limited set of query functions when matching MQTs to user specified queries. The user-specified query and the MQT query must both be supported by the SQE optimizer.

The supported function in the MQT query by the MQT matching algorithm includes:

- Single table and join queries
- WHERE clause
- GROUP BY and optional HAVING clauses
- ORDER BY
- FETCH FIRST n ROWS
- Views, common table expressions, and nested table expressions
- UNIONs
- Partitioned tables

There is limited support in the MQT matching algorithm for the following:

- Scalar subselects
- User Defined Functions (UDFs) and user-defined table functions

- Recursive Common Table Expressions (RCTE)
- The following scalar functions:
  - ATAN2
  - DAYNAME
  - DBPARTITIONNAME
  - DECRYPT_BIT
  - DECRYPT_BINARY
  - DECRYPT_CHAR
  - DECRYPT_DB
  - DIFFERENCE
  - DLVALUE
  - DLURLPATH
  - DLURLPATHONLY
  - DLURLSEVER
  - DLURLSCHEME
  - DLURLCOMPLETE
  - ENCRYPT_AES
  - ENCRYPT_RC2
  - ENCRYPT_TDES
  - GENERATE_UNIQUE
  - GETHINT
  - IDENTITY_VAL_LOCAL
  - INSERT
  - MONTHNAME
  - MONTHS_BETWEEN
  - NEXT_DAY
  - RAND
  - RAISE_ERROR
  - REPEAT
  - REPLACE
  - ROUND_TIMESTAMP
  - SOUNDEX
  - TIMESTAMP_FORMAT
  - TIMESTAMPDIFF
  - TRUNC_TIMESTAMP
  - VARCHAR_FORMAT
  - WEEK_ISO

It is recommended that the MQT only contain references to columns and column functions. In many environments, queries that contain constants have the constants converted to parameter markers. This conversion allows a much higher degree of ODP reuse. The MQT matching algorithm attempts to match constants in the MQT with parameter markers or host variable values in the query. However, in some complex cases this support is limited and could result in the MQT not matching the query.

Related concepts
Query dispatcher

The function of the dispatcher is to route the query request to either CQE or SQE, depending on the attributes of the query. All queries are processed by the dispatcher. It cannot be bypassed.

Related reference

Details on the MQT matching algorithm
What follows is a generalized discussion of how the MQT matching algorithm works.

Using MQTs during query optimization

Before using MQTs, you need to consider your environment attributes.

To even consider using MQTs during optimization the following environmental attributes must be true:

- The query must specify ALWCPYDTA(*OPTMIZE) or INSENSITIVE cursor.
- The query must not be a SENSITIVE cursor.
- The table to be replaced with an MQT must not be update or delete capable for this query.
- The MQT currently has the ENABLE QUERY OPTIMIZATION attribute active
- The MATERIALIZED_QUERY_TABLE_USAGE QAQQINI option must be set to *ALL or *USER to enable use of MQTs. The default setting of MATERIALIZED_QUERY_TABLE_USAGE does not allow usage of MQTs.
- The timestamp of the last REFRESH TABLE for an MQT is within the duration specified by the MATERIALIZED_QUERY_TABLE_REFRESH_AGE QAQQINI option. Or *ANY is specified, which allows MQTs to be considered regardless of the last REFRESH TABLE. The default setting of MATERIALIZED_QUERY_TABLE_REFRESH_AGE does not allow usage of MQTs.
- The query must be run through SQE.
- The following QAQQINI options must match: IGNORE_LIKE_REDUNDANT_SHIFTS, NORMALIZE_DATA, and VARIABLE_LENGTH_OPTIMIZATION. These options are stored at CREATE materialized query table time and must match the options specified at query run time.
- The commit level of the MQT must be greater than or equal to the query commit level. The commit level of the MQT is either specified in the MQT query using the WITH clause. Or it is defaulted to the commit level that the MQT was run under when it was created.
- The field procedure encoded comparison (QAQQINI FIELDPROC_ENCODED_COMPARISON option) level of the MQT must be greater than or equal to the query specified field procedure encoded comparison level.

MQT examples

The following are examples of using MQTs.

Example 1

The first example is a query that returns information about employees whose job is DESIGNER. The original query:

```
SELECT D.deptname, D.location, E.firstnme, E.lastname, E.salary+E.comm+E.bonus as total_sal
FROM Department D, Employee E
WHERE D.deptno=E.workdept
AND E.job = 'DESIGNER'
```

Create a table, MQT1, that uses this query:

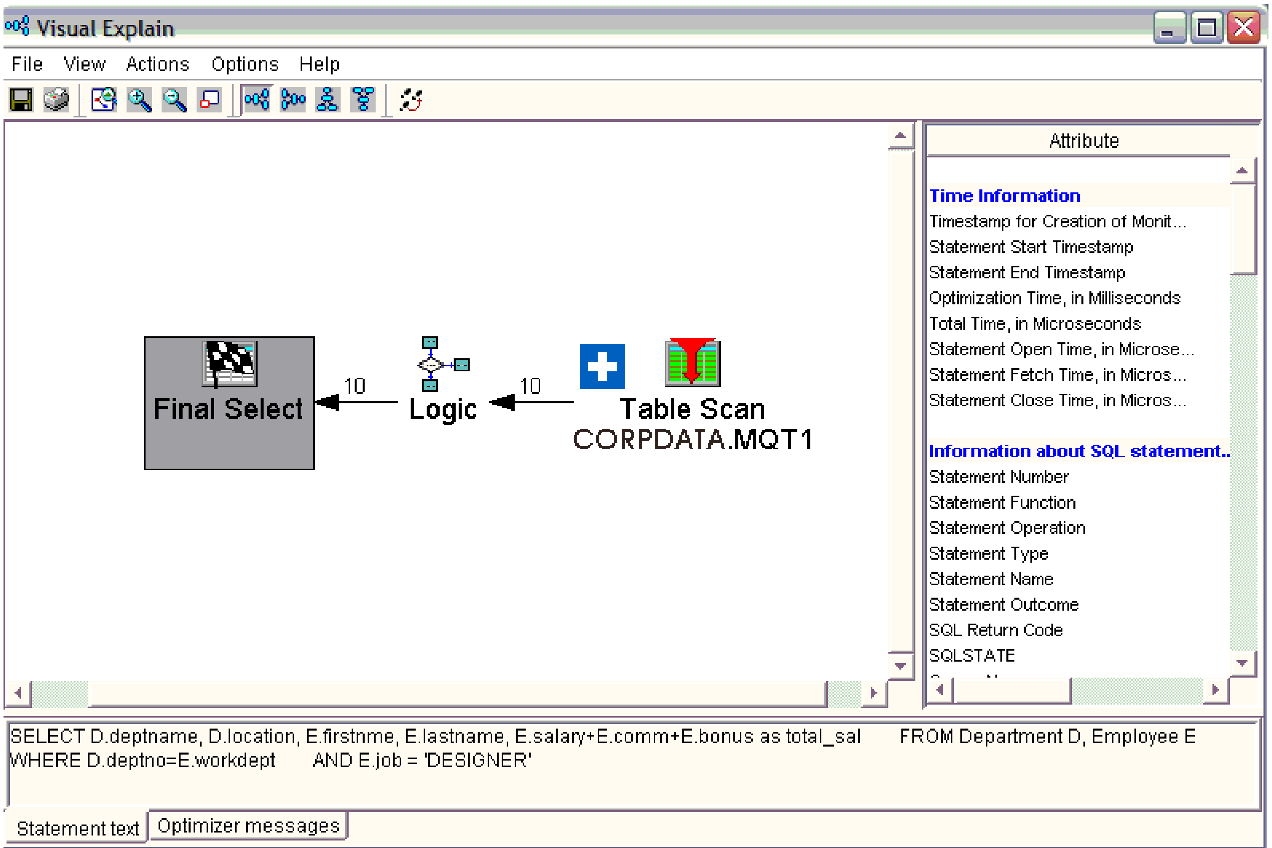
```
CREATE TABLE MQT1
       AS (SELECT D.deptname, D.location, E.firstnme, E.lastname, E.salary, E.comm, E.bonus,
E.job
FROM Department D, Employee E
WHERE D.deptno=E.workdept)
DATA INITIALLY IMMEDIATE  REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER
```

Resulting new query after replacing the specified tables with the MQT.

```
SELECT M.deptname, M.location, M.firstnme, M.lastname, M.salary+M.comm+M.bonus as total_sal
FROM MQT1 M
WHERE M.job = 'DESIGNER'
```

In this query, the MQT matches part of the user query. The MQT is placed in the FROM clause and replaces tables DEPARTMENT and EMPLOYEE. Any remaining selection not done by the MQT query (M.job= 'DESIGNER') is done to remove the extra rows. The result expression, M.salary+M.comm +M.bonus, is calculated. JOB must be in the select-list of the MQT so that the additional selection can be performed.

Visual Explain diagram of the query when using the MQT:



Example 2

Get the total salary for all departments that are located in 'NY'. The original query:

```
SELECT D.deptname, sum(E.salary)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept AND D.location = 'NY'
GROUP BY D.deptname
```

Create a table, MQT2, that uses this query:

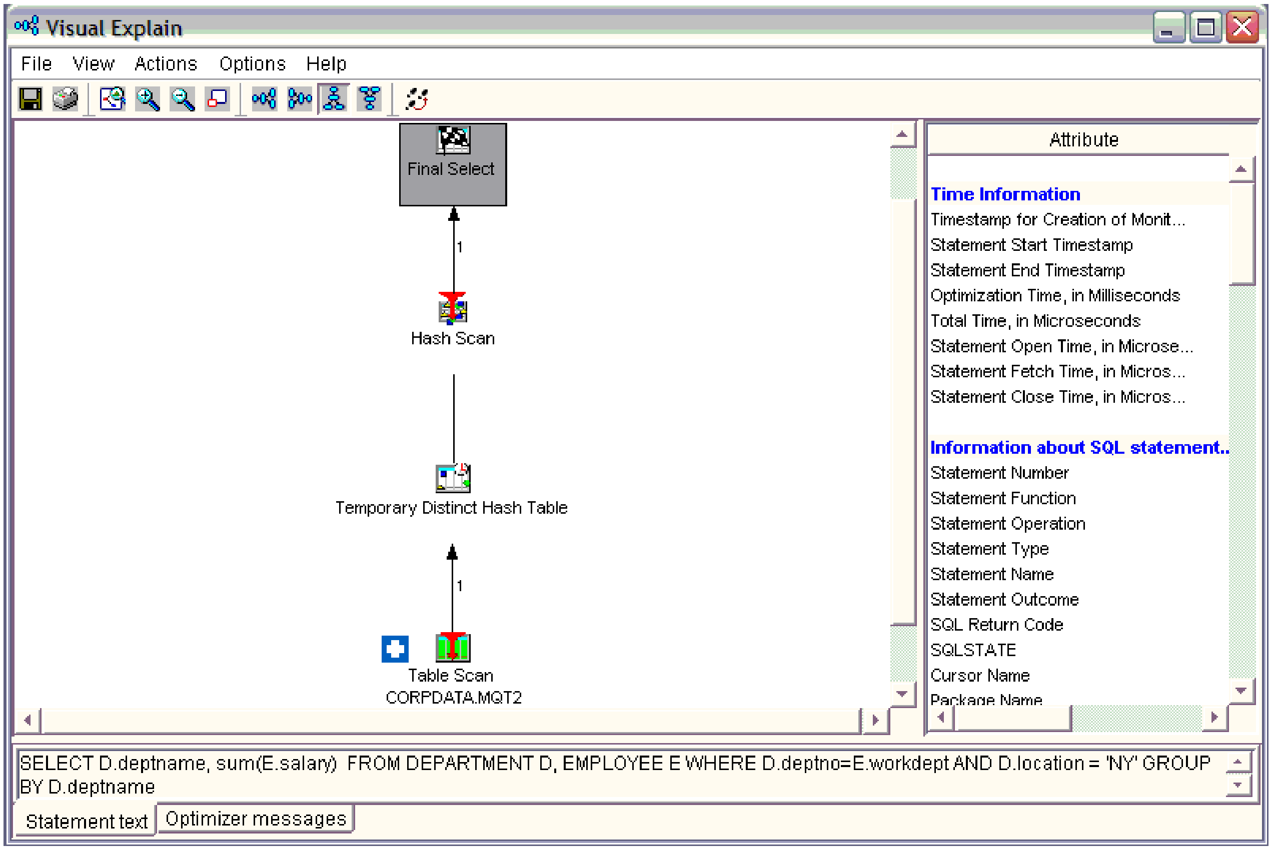
```
CREATE TABLE MQT2
        AS (SELECT D.deptname, D.location, sum(E.salary) as sum_sal
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept
GROUP BY D.Deptname, D.location)
DATA INITIALLY IMMEDIATE  REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER
```

Resulting new query after replacing the specified tables with the MQT:

```
SELECT M.deptname, sum(M.sum_sal)
FROM  MQT2 M
WHERE  M.location = 'NY'
GROUP BY M.deptname
```

Since the MQT could potentially produce more groups than the original query, the final resulting query must group again and SUM the results to return the correct answer. Also, the selection M.location='NY' must be part of the new query.

Visual Explain diagram of the query when using the MQT:



Details on the MQT matching algorithm

What follows is a generalized discussion of how the MQT matching algorithm works.

The tables specified in the query and the MQT are examined. If the MQT and the query specify the same tables, then the MQT can potentially be used and matching continues. If the MQT references tables not referenced in the query, then the unreferenced table is examined to determine if it is a parent table in referential integrity constraint. If the foreign key is non-nullable and the two tables are joined using a primary key or foreign key equal predicate, then the MQT can still be potentially used.

Example 3

The MQT contains fewer tables than the query:

```
SELECT D.deptname, p.projname, sum(E.salary)
FROM DEPARTMENT D, EMPLOYEE E, EMPPROJACT EP,  PROJECT P
WHERE D.deptno=E.workdept AND E.Empno=ep.empno
AND ep.projno=p.projno
GROUP BY D.DEPTNAME, p.projname
```

Create an MQT based on the preceding query:

```
CREATE TABLE MQT3
        AS (SELECT D.deptname,  sum(E.salary) as sum_sal, e.workdept, e.empno
```

```
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept
GROUP BY D.Deptname, e.workdept, e.empno)
DATA INITIALLY IMMEDIATE  REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER
```

The rewritten query:

```
SELECT M.deptname, p.projname, SUM(M.sum_sal)
FROM MQT3 M, EMPPROJACT EP,  PROJECT P
WHERE M.Empno=ep.empno AND ep.projno=p.projno
GROUP BY M.deptname, p.projname
```

All predicates specified in the MQT, must also be specified in the query. The query could contain additional predicates. Predicates specified in the MQT must match exactly the predicates in the query. Any additional predicates specified in the query, but not in the MQT must be able to be derived from columns projected from the MQT. See previous example 1.

Example 4

Set the total salary for all departments that are located in 'NY'.

```
SELECT D.deptname, sum(E.salary)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept AND D.location =  ?
GROUP BY D.Deptname
```

Create an MQT based on the preceding query:

```
CREATE TABLE MQT4
        AS (SELECT D.deptname, D.location, sum(E.salary) as sum_sal
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept AND D.location = 'NY'
GROUP BY D.deptnamet, D.location)
DATA INITIALLY IMMEDIATE  REFRESH DEFERRED
    ENABLE QUERY OPTIMIZATION
    MAINTAINED BY USER
```

In this example, the constant 'NY' was replaced by a parameter marker and the MQT also had the local selection of location='NY' applied to it when the MQT was populated. The MQT matching algorithm matches the parameter marker and to the constant 'NY' in the predicate D.Location=?. It verifies that the values of the parameter marker are the same as the constant in the MQT; therefore the MQT can be used.

The MQT matching algorithm also attempts to match where the predicates between the MQT and the query are not the same. For example, if the MQT has a predicate SALARY > 50000, and the query has the predicate SALARY > 70000, the MQT contains the rows necessary to run the query. The MQT is used in the query, but the predicate SALARY > 70000 is left as selection in the query, so SALARY must be a column of the MQT.

Example 5

```
SELECT D.deptname, sum(E.salary)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept AND D.location =  'NY'
GROUP BY D.deptname
```

Create an MQT based on the preceding query:

```
CREATE TABLE MQT5
        AS (SELECT D.deptname, E.salary
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept)
DATA INITIALLY IMMEDIATE  REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER
```

In this example, since D.Location is not a column of the MQT, the user query local selection predicate Location='NY' cannot be determined, so the MQT cannot be used.

Example 6

```
SELECT D.deptname, sum(E.salary)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.deptno=E.workdept
GROUP BY D.deptname
```

Create an MQT based on the preceding query:

```
CREATE TABLE MQT6(workdept, sumSalary)
AS (SELECT workdept, sum(salary)
    FROM EMPLOYEE
    GROUP BY workdept )
DATA INITIALLY IMMEDIATE REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER
```

In this example, the SUM(salary) aggregation is pushed down through the join to the EMPLOYEE table, allowing for a match and substitution of MQT6. A regrouping to (sum(sum(salary))) is defined at the top of the query to compensate for the grouping pushdown.

Instead of department joining to all the rows in the employee table, it now has the opportunity to join to the predetermined aggregates in MQT6. This type of MQT substitution can result in significant reduction of processing and IO.

If the MQT contains grouping, then the query must be a grouping query. The simplest case is where the MQT and the query specify the same list of grouping columns and column functions.

In some cases, if the MQT specifies group by columns that are a superset of query group by columns, the query can be rewritten to do regrouping. This regrouping reaggregates the groups of the MQT into the groups required by the query. When regrouping is required, the column functions need to be recomputed. The following table shows the supported regroup expressions.

The regrouping expression/aggregation rules are:

Table 28. Expression/aggregation rules for MQTs

Query	**MQT**	**Final query**
COUNT(*)	COUNT(*) as cnt	SUM(cnt)
COUNT(*)	COUNT(C2) as cnt2 (where c2 is non-nullable)	SUM(cnt2)
COUNT(c1)	COUNT(c1) as cnt	SUM(cnt)
COUNT(C1) (where C1 is non-nullable)	COUNT(C2) as cnt2 (where C2 is non-nullable)	SUM(cnt2)
COUNT(distinct C1)	C1 as group_c1 (where C1 is a grouping column)	COUNT(group_C1)
COUNT(distinct C1)	where C1 is not a grouping column	MQT not usable
COUNT(C2) where C2 is from a table not in the MQT	COUNT(*) as cnt	cnt*COUNT(C2)
COUNT(distinct C2) where C2 is from a table not in the MQT	Not applicable	COUNT(distinct C2)
SUM(C1)	SUM(C1) as sm	SUM(sm)

Table 28. Expression/aggregation rules for MQTs (continued)		
Query	**MQT**	**Final query**
SUM(C1)	C1 as group_c1, COUNT(*) as cnt (where C1 is a grouping column)	SUM(group_c1 * cnt)
SUM(C2) where C2 is from a table not in the MQT	COUNT(*) as cnt	cnt*SUM(C2)
SUM(distinct C1)	C1 as group_c1 (where C1 is a grouping column)	SUM(group_C1)
SUM(distinct C1)	where C1 is not a grouping column	MQT not usable
SUM(distinct C2) where C2 is from a table not in the MQT	Not applicable	SUM(distinct C2)
MAX(C1)	MAX(C1) as mx	MAX(mx)
MAX(C1)	C1 as group_C1 (where C1 is a grouping column)	MAX(group_c1)
MAX(C2) where C2 is from a table not in the MQT	Not applicable	MAX(C2)
MIN(C1)	MIN(C1) as mn	MIN(mn)
MIN(C1)	C1 as group_C1 (where C1 is a grouping column)	MIN(group_c1)
MIN(C2) where C2 is from a table not in the MQT	Not applicable	MIN(C2)
GROUPING(C1)	GROUPING(C1) as grp	grp
GROUPING(C2) where C2 is from a table not in the MQT	Not applicable	GROUPING(C2)

MQT matching does not support ARRAY_AGG, XMLAGG, and XMLGROUP grouping functions. AVG, STDDEV, STDDEV_SAMP, VARIANCE_SAMPand VAR_POP are calculated using combinations of COUNT and SUM. If AVG, STDDEV, or VAR_POP are included in the MQT and regroup requires recalculation of these functions, the MQT cannot be used. It is recommended that the MQT only use COUNT, SUM, MIN, and MAX. If the query contains AVG, STDDEV, or VAR_POP, it can be recalculated using COUNT and SUM.

If FETCH FIRST N ROWS is specified in the MQT, then FETCH FIRST N ROWS must also be specified in the query. Also, the number of rows specified for the MQT must be greater than or equal to the number of rows specified in the query. It is not recommended that an MQT contain the FETCH FIRST N ROWS clause.

The ORDER BY clause on the MQT can be used to order the data in the MQT if a REFRESH TABLE is run. It is ignored during MQT matching and if the query contains an ORDER BY clause, it is part of the rewritten query.

Related reference

MQT supported function

Although an MQT can contain almost any query, the optimizer only supports a limited set of query functions when matching MQTs to user specified queries. The user-specified query and the MQT query must both be supported by the SQE optimizer.

Determining unnecessary MQTs

You can easily determine which MQTs are being used for query optimization. However, you can now easily find all MQTs and retrieve statistics on MQT usage as a result of System i Navigator and IBM i functionality.

To assist you in tuning your performance, this function produces statistics on MQT usage in a query. To access through the System i Navigator, navigate to: **Database** > **Schemas** > **Tables**. Right-click your table and select **Show Materialized Query Tables**. You can also view MQT usage information by right-click on Tables or Views folder and select **Show Materialized Query Tables**. This action displays usage information for MQTs created over all the tables or view in that schema.

Note: You can also view the statistics through an application programming interface (API).

In addition to all existing attributes of an MQT, two fields can help you determine unnecessary MQTs.

These fields are:

Last Query Use
: States the timestamp when the MQT was last used by the optimizer to replace user specified tables in a query.

Query Use Count
: Lists the number of instances the MQT was used by the optimizer to replace user specified tables in a query.

The fields start and stop counting based on your situation, or the actions you are currently performing on your system. A save and restore procedure does not reset the statistics counter if the MQT is restored over an existing MQT. If an MQT is restored that does not exist on the system, the statistics are reset.

Related information

Retrieve member description (QUSRMBRD) command

Summary of MQT query recommendations

Follow these recommendations when using MQT queries.

- Do not include local selection or constants in the MQT because that limits the number of user-specified queries where the optimizer can use the MQT.
- For grouping MQTs, only use the SUM, COUNT, MIN, and MAX grouping functions. The query optimizer can recalculate AVG, STDDEV, and VAR_POP in user specified queries.
- Specifying FETCH FIRST N ROWS in the MQT limits the number of user-specified queries where the query optimizer can use the MQT. Not recommended.
- If the MQT is created with DATA INITIALLY DEFERRED, consider specifying DISABLE QUERY OPTIMIZATION to prevent the optimizer from using the MQT until it has been populated. When the MQT is populated and ready for use, the ALTER TABLE statement with ENABLE QUERY OPTIMIZATION enables the MQT.

In addition, consider using a sparse index or EVI INCLUDE additional aggregates rather than an MQT if you are concerned with stale data.

MQT tables need to be optimized just like non-MQT tables. It is recommended that indexes are created over the MQT columns used for selection, join, and grouping, as appropriate. Column statistics are collected for MQT tables.

The database monitor shows the list of MQTs considered during optimization. This information is in the 3030 record. If MQTs have been enabled through the QAQQINI file, and an MQT exists over at least one of the tables in the query, there is a 3030 record for the query. Each MQT has a reason code indicating that it was used or if it was not used, why it was not used.

Related concepts

How the EVI works

EVIs work in different ways for costing and implementation.

Related reference

Sparse index optimization

An SQL sparse index is like a select/omit access path. Both the sparse index and the select/omit logical file contain only keys that meet the selection specified. For a sparse index, the selection is specified with a WHERE clause. For a select/omit logical file, the selection is specified in the DDS using the COMP operation.

Recursive query optimization

Certain applications and data are recursive by nature. Examples of such applications are a bill-of-material, reservation, trip planner, or networking planning system. Data in one results row has a natural relationship (call it a parent, child relationship) with data in another row or rows. The kinds of recursion implemented in these systems can be performed by using SQL Stored Procedures and temporary results tables. However, the use of a recursive query to facilitate the access of this hierarchical data can lead to a more elegant and better performing application.

Recursive queries can be implemented by defining either a Recursive Common Table Expression (RCTE) or a Recursive View.

Recursive query example

A recursive query is one that is defined by a Union All with an initialization fullselect that seeds the recursion. The iterative fullselect contains a direct reference to itself in the FROM clause.

There are additional restrictions as to what can be specified in the definition of a recursive query. Those restrictions can be found in SQL Programming topic.

Functions like grouping, aggregation, or distinct require a materialization of all the qualifying records before performing the function. These functions cannot be allowed within the iterative fullselect itself. The functions must be placed in the main query, allowing the recursion to complete.

The following is an example of a recursive query over a table called flights, that contains information about departure and arrival cities. The query returns all the flight destinations available by recursion from the two specified cities (New York and Chicago). It also returns the number of connections and total cost to arrive at that final destination.

This example uses the recursion process to also accumulate information like the running cost and number of connections. Four values are put in the queue entry. These values are:

- The originating departure city (either Chicago or New York) because it remains fixed from the start of the recursion
- The arrival city which is used for subsequent joins
- The incrementing connection count
- The accumulating total cost to reach each destination

Typically the data needed for the queue entry is less than the full record (sometimes much less) although that is not the case for this example.

```
CREATE TABLE flights
    (
        departure CHAR (10) NOT NULL WITH DEFAULT,
        arrival CHAR (10) NOT NULL WITH DEFAULT,
        carrier CHAR (15) NOT NULL WITH DEFAULT,
        flight_num CHAR (5) NOT NULL WITH DEFAULT,
        ticket INT NOT NULL WITH DEFAULT)

WITH destinations (departure, arrival, connects, cost ) AS
  (
  SELECT  f.departure,f.arrival, 0, ticket
  FROM flights f
  WHERE f.departure = 'Chicago' OR
      f.departure = 'New York'
  UNION ALL
  SELECT
      r.departure, b.arrival, r.connects + 1,
```

```
      r.cost + b.ticket
  FROM  destinations r, flights b
  WHERE r.arrival = b.departure
  )
SELECT DISTINCT departure, arrival, connects, cost
FROM destinations
```

The following is the initialization fullselect of the preceding query. It seeds the rows that start the recursion process. It provides the initial destinations (arrival cities) that are a direct flight from Chicago or New York.

```
SELECT  f.departure,f.arrival, 0, ticket
FROM flights f
WHERE f.departure='Chicago' OR
          f.departure='New York'
```

The following is the iterative fullselect of the preceding query. It contains a single reference in the FROM clause to the destination recursive common table expression. It also sources further recursive joins to the same flights table. The arrival values of the parent row (initially direct flights from New York or Chicago) are joined with the departure value of the subsequent child rows. It is important to identify the correct parent/child relationship on the recursive join predicate or infinite recursion can occur. Other local predicates can also be used to limit the recursion. For example, for a limit of at most 3 connecting flights, a local predicate using the accumulating connection count, r.connects<=3, can be specified.

```
SELECT
  r.departure, b.arrival, r.connects + 1 ,
  r.cost + b.ticket
FROM  destinations r, flights b
WHERE r.arrival=b.departure
```

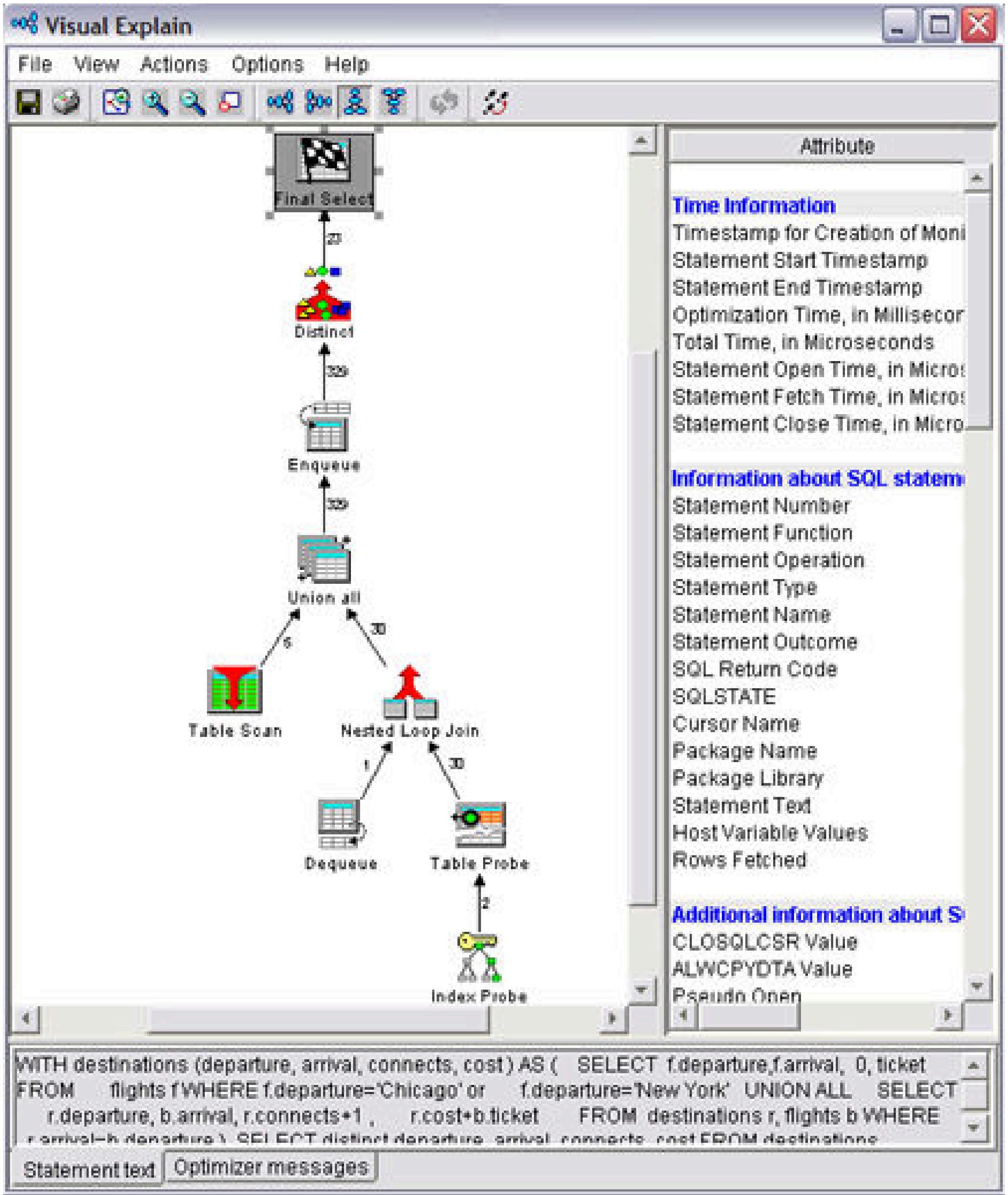
The main query is the query that references the recursive common table expression or view. It is in the main query where requests like grouping, ordering, and distinct are specified.

```
SELECT DISTINCT departure, arrival, connects, cost
FROM destinations
```

Implementation considerations

To implement a source for the recursion, a new temporary data object is provided called a queue. As rows meet the requirements of either the initialization fullselect or the iterative fullselect, they are pulled up through the union all. Values necessary to feed the continuing recursion process are captured and placed in an entry on the queue: an enqueue operation.

At query runtime, the queue data source then takes the place of the recursive reference in the common table expression or view. The iterative fullselect processing ends when the queue is exhausted of entries or a fetch N rows limitation has been met. The recursive queue feeds the recursion process and holds transient data. The join between dequeuing of these queue entries and the rest of the fullselect tables is always a constrained join, with the queue on the left.



Multiple initialization and iterative fullselects

The use of multiple initialization and iterative fullselects specified in the recursive query definition allows for a multitude of data sources and separate selection requirements to feed the recursion process.

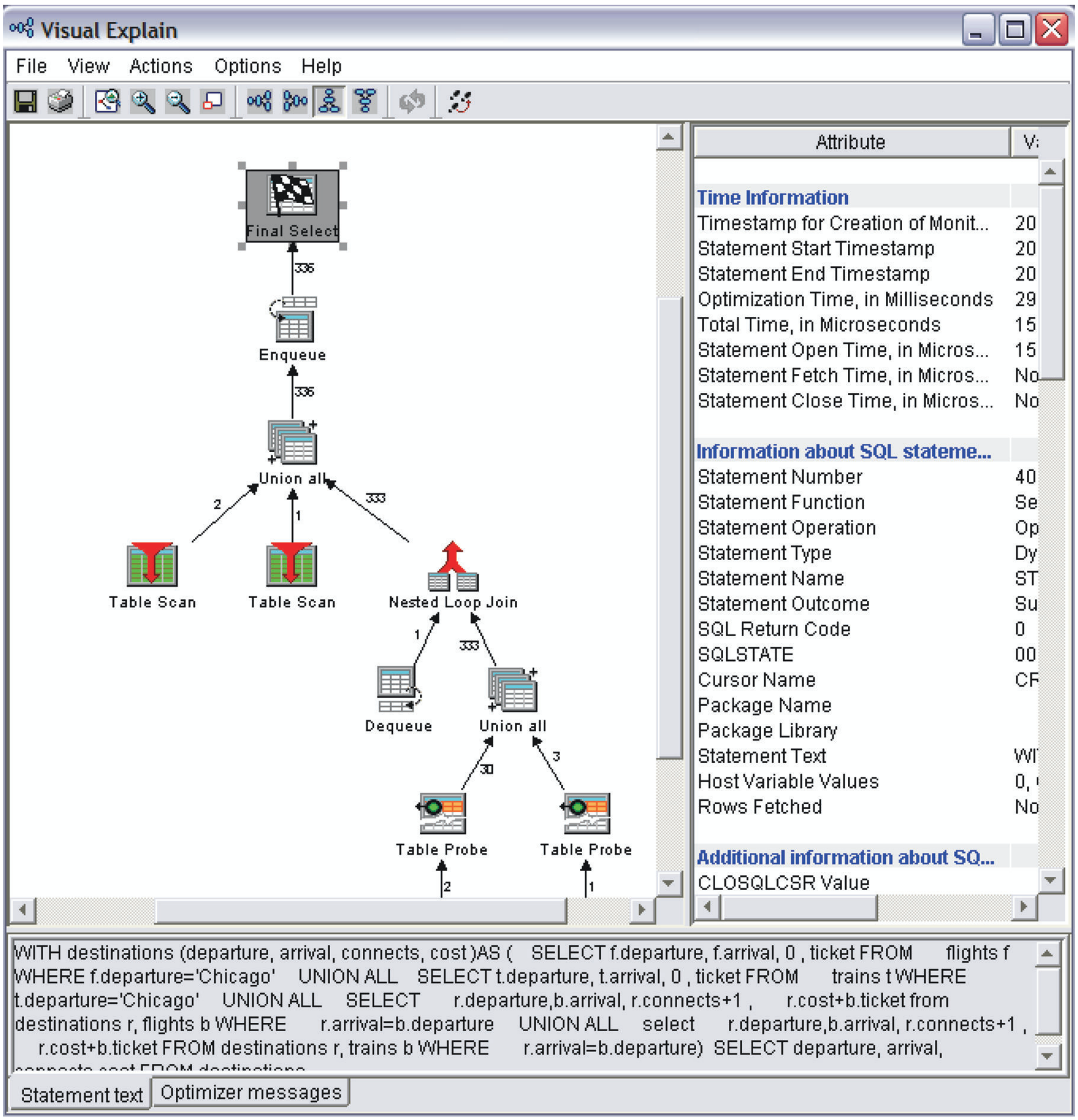
For example, the following query allows for final destinations accessible from Chicago by both flight and train travel.

```
WITH destinations (departure, arrival, connects, cost ) AS
(
  SELECT f.departure, f.arrival, 0 , ticket
  FROM flights f
  WHERE f.departure='Chicago'
  UNION ALL
  SELECT t.departure, t.arrival, 0 , ticket
    FROM trains t
    WHERE t.departure='Chicago'
  UNION ALL
```

```
  SELECT
        r.departure,b.arrival, r.connects + 1 ,
        r.cost + b.ticket
    FROM destinations r, flights b
    WHERE r.arrival=b.departure
  UNION ALL
    SELECT
            r.departure,b.arrival, r.connects+1 ,
            r.cost+b.ticket
    FROM destinations r, trains b
    WHERE r.arrival=b.departure)

SELECT departure, arrival, connects,cost
FROM destinations;
```

All rows coming out of the RCTE/View are part of the recursion process and need to be fed back in. When there are multiple fullselects referencing the common table expression, the query is rewritten by the optimizer to process all non-recursive initialization fullselects first. Then, using a single queue feed, those same rows and all other row results are sent equally to the remaining iterative fullselects. No matter how you order the initialization and iterative fullselects in the definition of the RCTE/view, the initialization fullselects run first. The iterative fullselects share equal access to the contents of the queue.



Predicate pushing

When processing most queries with non-recursive common table expressions or views, local predicates specified on the main query are pushed down so fewer records need to be materialized. Pushing local predicates from the main query into the defined recursive part of the query (through the Union ALL), however, could considerably alter the process of recursion itself. So as a rule, the Union All specified in a recursive query is currently a predicate fence. Predicates are not pushed down or up, through this fence.

The following is an example of how pushing a predicate in to the recursion limits the recursive results and alter the intent of the query.

The intent of the query is to find all destinations accessible from 'Chicago', not including the final destination of 'Dallas'. Pushing the "arrival<>'Dallas'" predicate into the recursive query alters the output of the intended results. It prevents the output of final destinations where 'Dallas' was an intermediate stop.

```
WITH destinations (departure, arrival, connects, cost ) AS
(
  SELECT f.departure,f.arrival, 0, ticket
  FROM flights f
  WHERE f.departure='Chicago'
  UNION ALL
  SELECT
            r.departure, b.arrival, r.connects + 1 ,
            r.cost + b.ticket
  FROM  destinations r, flights b
  WHERE r.arrival=b.departure
)
SELECT departure, arrival, connects, cost
FROM destinations
WHERE arrival != 'Dallas'
```

Conversely, the following is an example where a local predicate applied to all the recursive results is a good predicate to put in the body of the recursive definition because it could greatly decrease the number of rows materialized from the RCTE/View. The better query request here is to specify the r.connects <=3 local predicate with in the RCTE definition, in the iterative fullselect.

```
WITH destinations (departure, arrival, connects, cost ) AS
(
  SELECT f.departure,f.arrival, 0, ticket
  FROM flights f
  WHERE f.departure='Chicago' OR
      f.departure='New York'
  UNION ALL
  SELECT
            r.departure, b.arrival, r.connects + 1 ,
            r.cost + b.ticket
  FROM  destinations r, flights b
    WHERE r.arrival=b.departure
)
SELECT departure, arrival, connects, cost
FROM destinations
WHERE r.connects<=3
```

Placement of local predicates is key in recursive queries. They can incorrectly alter the recursive results if pushed into a recursive definition. Or they can cause unnecessary rows to be materialized and then rejected, when a local predicate could legitimately help limit the recursion.

Specifying SEARCH consideration

Certain applications dealing with hierarchical, recursive data could have a requirement in how data is processed: by depth or by breadth.

Using a queuing (First In First Out) mechanism to track the recursive join key values implies the results are retrieved in breadth first order. Breadth first means retrieving all the direct children of a parent row before retrieving any of the grandchildren of that same row. This retrieval is an implementation distinction, however, and not a guarantee.

Applications might want to guarantee how the data is retrieved. Some applications might want to retrieve the hierarchical data in depth first order. Depth first means that all the descendents of each immediate child row are retrieved before the descendents of the next child are retrieved.

The SQL architecture allows for the guaranteed specification of how the application retrieves the resulting data by the use of the SEARCH DEPTH FIRST or BREADTH FIRST keyword. When this option is specified, name the recursive join value, identify a set sequence column, and provide the sequence column in an outer ORDER BY clause. The results are output in depth or breadth first order. Note this ordering is ultimately a relationship sort and not a value-based sort.

Here is the preceding example output in depth first order.

```
WITH destinations (departure, arrival, connects, cost ) AS
(
  SELECT f.departure, f.arrival, 0 , ticket
  FROM flights f
  WHERE f.departure='Chicago' OR f.departure='New York'
  UNION ALL
  SELECT
            r.departure,b.arrival, r.connects+1 ,
            r.cost+b.ticket
    FROM  destinations r, flights b
    WHERE r.arrival=b.departure)

SEARCH DEPTH FIRST BY arrival SET depth_sequence

    SELECT *
    FROM destinations
    ORDER BY depth_sequence
```

If the ORDER BY clause is not specified in the main query, the sequencing option is ignored. To facilitate the correct sort there is additional information put on the queue entry during recursion. With BREADTH FIRST, it is the recursion level number and the immediate ancestor join value, so sibling rows can be sorted together. A depth first search is a little more data intensive. With DEPTH FIRST, the query engine needs to represent the entire ancestry of join values leading up to the current row and put that information in a queue entry. Also, because these sort values are not coming from an external data source, the sort implementation is always a temporary sorted list (no indexes possible).

Do not use the SEARCH option if you do not need your data materialized in a depth or breadth first manner. There is additional CPU and memory overhead to manage the sequencing information.

Specifying CYCLE considerations

Recognizing that data in the tables used in a recursive query might be cyclic in nature is important to preventing infinite loops.

The SQL architecture allows for the optional checking for cyclic data and discontinuing the repeating cycles at that point. This additional checking is done by the use of the CYCLE option. The correct join recursion value must be specified on the CYCLE request and a cyclic indicator must be specified. The cyclic indicator could be optionally output in the main query and can be used to help determine and correct errant cyclic data.

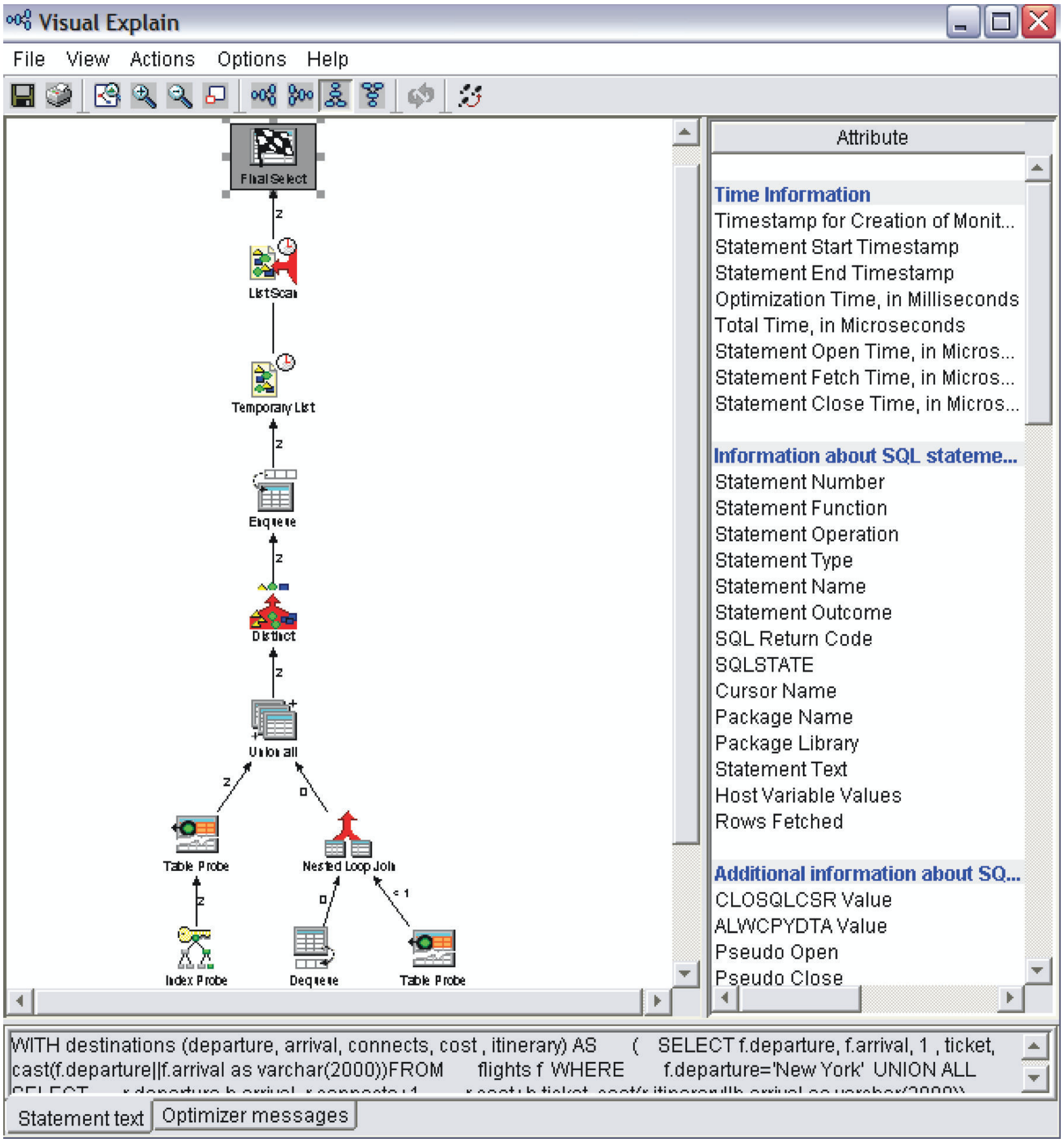
```
WITH destinations (departure, arrival, connects, cost , itinerary) AS
  (
  SELECT f.departure, f.arrival, 1 , ticket, CAST(f.departure||f.arrival AS VARCHAR(2000))
  FROM flights f
  WHERE f.departure='New York'
  UNION ALL
  SELECT r.departure,b.arrival, r.connects+1 ,
      r.cost+b.ticket, cast(r.itinerary||b.arrival AS varchar(2000))
    FROM destinations r, flights b
    WHERE r.arrival = b.departure)
CYCLE arrival SET cyclic TO '1' DEFAULT '0' USING Cycle_Path

SELECT departure, arrival, itinerary, cyclic
FROM destinations
```

When a cycle is determined to be repeating, the output of that cyclic sequence of rows is stopped. To check for a 'repeated' value however, the query engine needs to represent the entire ancestry of the join values leading up to the current row in order to look for the repeating join value. This ancestral history is information that is appended to with each recursive cycle and put in a field on the queue entry.

To implement this history field, the query engine uses a compressed representation of the recursion values on the ancestry chain. The query engine can then do a fixed length, quicker scan through the accumulating ancestry to determine if the value has been seen before. This compressed representation is determined by the use of a distinct node in the query tree.

Do not use the CYCLE option unless you know your data is cyclic, or you want to use it specifically to help find the cycles for correction or verification purposes. There is additional CPU and memory overhead to manage and check for repeating cycles before a given row is materialized.



SMP and recursive queries

Recursive queries can benefit as much from symmetric multiprocessing (SMP) as do other queries on the system.

Recursive queries and parallelism, however, present some unique requirements. The initialization fullselect of a recursive query is the fullselect that seeds the initial values of the recursion. It is likely to produce only a small fraction of the ultimate results that cycle through the recursion process. The query optimizer does not want each of the threads running in parallel to have a unique queue object that feeds only itself. This results in some threads having way too much work to do and others threads quickly depleting their work.

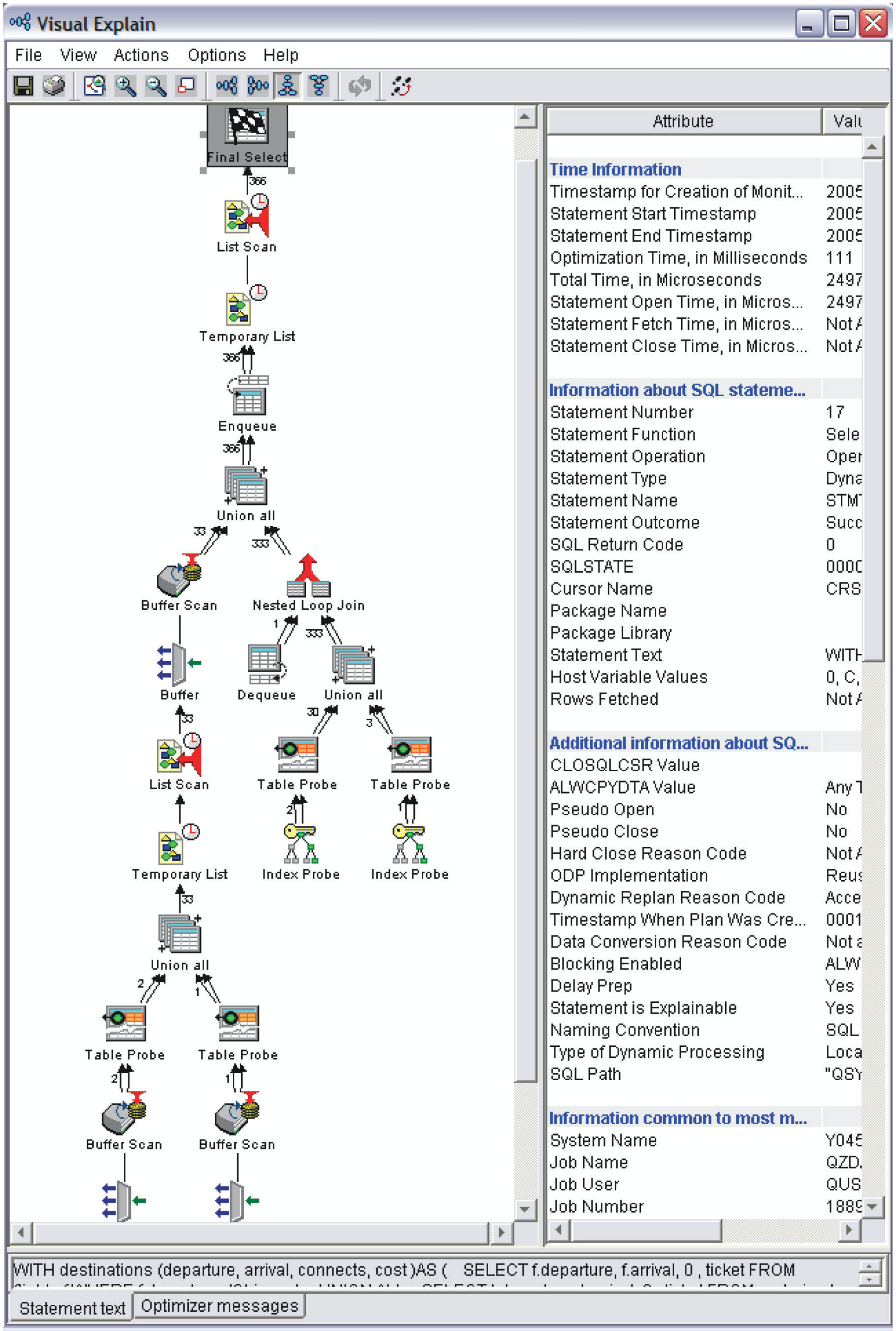
The best way to handle this work is to have all the threads share the same queue. This method allows a thread to enqueue a new recursive key value just as a waiting thread is there to dequeue that request. A shared queue allows all threads to actively contribute to the overall depletion of the queue entries until no thread is able to contribute more results.

Having multiple threads share the same queue, however, requires some management by the Query runtime so that threads do not prematurely end. Some buffering of the initial seed values might be necessary. This buffering is illustrated in the following query, where there are two fullselects that seed the recursion. A buffer is provided so that no thread hits a dequeue state and terminates before the query has seeded enough recursive values to get things going.

The following Visual Explain diagram shows the plan for the following query run with `CHGQRYA DEGREE(*NBRTASKS 4)`. It shows how the results of the multiple initialization fullselects are buffered up. The multiple threads, illustrated by the multiple arrow lines, are acting on the enqueue and dequeue request nodes. As with all SMP queries, the multiple threads, in this case 4, put their results into a Temporary List object which becomes the output for the main query.

```
cl:chgqrya degree(*nbrtasks 4);

WITH destinations (departure, arrival, connects, cost )AS
 (
  SELECT f.departure, f.arrival, 0 , ticket
  FROM flights f WHERE f.departure='Chicago'
  UNION ALL
  SELECT t.departure, t.arrival, 0 , ticket
    FROM trains t WHERE t.departure='Chicago'
  UNION ALL
  SELECT
      r.departure,b.arrival, r.connects+1 ,
      r.cost+b.ticket
    FROM destinations r, flights b
    WHERE r.arrival=b.departure
  UNION ALL
  SELECT
      r.departure,b.arrival, r.connects+1 ,
      r.cost+b.ticket
    FROM destinations r, trains b
    WHERE r.arrival=b.departure)
SELECT departure, arrival, connects,cost
FROM destinations;
```


Visual Explain
File View Actions Options Help
Final Select
List Scan
Temporary List
Enqueue
Union all
Buffer Scan
Nested Loop Join
Buffer
Dequeue
Union all
List Scan
Table Probe
Table Probe
Temporary List
Index Probe
Index Probe
Union all
Table Probe
Table Probe
Buffer Scan
Buffer Scan
Attribute
Valu
Time Information
Timestamp for Creation of Monit... 2005
Statement Start Timestamp 2005
Statement End Timestamp 2005
Optimization Time, in Milliseconds 111
Total Time, in Microseconds 2497
Statement Open Time, in Micros... 2497
Statement Fetch Time, in Micros... Not A
Statement Close Time, in Micros... Not A
Information about SQL stateme...
Statement Number 17
Statement Function Sele
Statement Operation Oper
Statement Type Dyna
Statement Name STMT
Statement Outcome Succ
SQL Return Code 0
SQLSTATE 0000
Cursor Name CRS
Package Name
Package Library
Statement Text WITH
Host Variable Values 0, C,
Rows Fetched Not A
Additional information about SQ...
CLOSQLCSR Value
ALWCPYDTA Value Any T
Pseudo Open No
Pseudo Close No
Hard Close Reason Code Not A
ODP Implementation Reus
Dynamic Replan Reason Code Acce
Timestamp When Plan Was Cre... 0001
Data Conversion Reason Code Not a
Blocking Enabled ALW
Delay Prep Yes
Statement is Explainable Yes
Naming Convention SQL
Type of Dynamic Processing Loca
SQL Path "QSY
Information common to most m...
System Name Y045
Job Name QZD
Job User QUS
Job Number 1889
WITH destinations (departure, arrival, connects, cost)AS (SELECT f.departure, f.arrival, 0 , ticket FROM
Statement text
Optimizer messages

Adaptive Query Processing

Adaptive Query Processing analyzes actual query run time statistics and uses that information for subsequent optimizations.

With rapidly increasing amounts of data, the price of miscalculating complex plans can result in dramatic performance problems. These problems might be measured in minutes or hours instead of seconds or minutes. Traditionally, optimizer architecture has attempted to overcome potential plan problems in several ways. The most common technique is to increase the amount of time spent optimizing a query, searching for safe alternatives. While additional time reduces the likelihood of a failed plan, it does not fundamentally avoid the problem.

The DB2 optimizer relies on statistical estimates to optimize a query. These estimates can be inaccurate for a number of reasons. The reasons include a lack of statistical metadata for the query tables, complex join conditions, skewed or rapidly changing data within the tables, and others.

The SQE query engine uses a technique called Adaptive Query Processing (AQP). AQP analyzes actual query run time statistics and uses that information to correct previous estimates. These updated estimates can provide better information for subsequent optimizations.

Related reference

Adaptive Query Processing in Visual Explain
You can use Visual Explain to request a new plan.

How AQP works

There are three main parts to AQP support.

- **Global Statistics Cache (GSC):** The “Global Statistics Cache” on page 9 is a system-side repository of statistical information gathered from actual query runs. When the SQE query engine observes a discrepancy between record count estimates and actual observed values, an entry might be made in the GSC. This entry provides the optimizer with more accurate statistical information for subsequent optimizations.
- **AQP Request Support:** This support runs after a query completes. The processing is done in a system task so it does not affect the performance of user applications. Estimated record counts are compared to the actual values. If significant discrepancies are noted, the AQP Request Support stores the observed statistic in the GSC. The AQP Request Support might also make specific recommendations for improving the query plan the next time the query runs.
- **AQP Handler:** The AQP Handler runs in a thread parallel to a running query and observes its progress. The AQP handler wakes up after a query runs for at least 2 seconds without returning any rows. Its job is to analyze the actual statistics from the partial query run, diagnose, and possibly recover from join order problems. These join order problems are due to inaccurate statistical estimates.

  The query can be reoptimized using partial observed statistics or specific join order recommendations or both. If this optimization results in a new plan, the old plan is terminated and the query restarted with the new plan, provided the query has not returned any results.

AQP looks for an unexpected starvation join condition when it analyzes join performance. Starvation join is a condition where a table late in the join order eliminates many records from the result set. In general, the query would perform better if the table that eliminates the large number of rows is first in the join order. When AQP identifies a table that causes an unexpected starvation join condition, the table is noted as the 'forced primary table'. The forced primary table is saved for a subsequent optimization of the query.

That subsequent optimization with the forced primary recommendation can be used in two ways:

- The forced primary table is placed first in the join order, overriding the join order implied by the statistical estimates. The rest of the join order is defined using existing techniques.
- The forced primary table can be used for LPG preselection against a large fact table in the join.

Related reference

Adaptive Query Processing in Visual Explain

You can use Visual Explain to request a new plan.

AQP example

Here is an example query with an explanation of how AQP could work.

```
SELECT * from t1, t2, t3, t4
WHERE t1.c1=t2.c1 AND t1.c2=t3.c2
AND t1.c3 = CURRENT DATE - t4.c3
AND t1.c5 < 50 AND t2.c6 > 40
AND t3.c7 < 100 AND t4.c8 - t4.c9 < 5
```

The WHERE clause of the preceding query contains a predicate, `t1.c3 = CURRENT DATE - t4.c3`, that is difficult to estimate. The estimation difficulty is due to the derivation applied to column `t4.c3` and the derivation involving columns `t4.c8` and `t4.c9`. For the purposes of this example, the predicate `t1.c3 = CURRENT DATE - t4.c3` actually eliminates all or nearly all records in the join.

Due to characteristics of the columns involved in that predicate, the statistical estimate has many rows returned from the join. The optimizer selects join order `t1, t3, t2, t4` based on the following record count estimates.

- Join `t1` to `t3` produces 33,000,000 rows.
- Join `t1, t3` result to `t2` produces 1,300,000 rows.
- Join `t1, t3, t2` result to `t4` (final result set) produces 5 million rows.

The join order is reasonable assuming that the final result set actually produces 5 million rows, but the estimate is incorrect. The query performs poorly since tables `t1, t3, t2` are joined first, producing 1,300,000 rows. These rows are all rejected by table `t4` and the `t1.c3 = CURRENT DATE - t4.c3` predicate (join starvation).

AQP identifies t4 as the forced primary table. The optimizer would choose `t1` as the second table in the join order since there are no join conditions between `t4` and `t2` or `t3`. Since the join condition between tables `t4` and `t1` selects few rows, this plan is likely many orders of magnitude faster than the original plan.

Related reference

Adaptive Query Processing in Visual Explain
You can use Visual Explain to request a new plan.

AQP join order

Adaptive Query Processing analyzes actual query run time join statistics and uses that information for subsequent join optimizations.

The SQE engine implements AQP join order recommendations in the following ways:

Subsequent to run

When each query completes, a fast check is done on key points of the query execution to compare actual selected records with the estimates. If there is a significant discrepancy, then a stand-alone task is notified to do a deeper analysis of the query execution.

The query plan and the execution statistics are passed to the task. A separate task is used for the in-depth analysis so the user job is not impacted while the deep analysis is done. Each step of the join is analyzed, looking for characteristics of starvation join. Starvation join shows a significant reduction in the number of rows produced compared to the previous step. The definition of what is considered significant depends on a number of factors.

If the criteria for starvation join are met, the actual number of records selected at key points of the query are compared to estimates. If there is a significant discrepancy between the actual and estimated record counts, the table at that join position is identified as a 'forced primary table'. This table is saved with the query plan in the system plan cache. When the query runs in the future, the optimizer retrieves the original plan from the system plan cache. The optimizer sees the forced primary table recommendation, and optimizes the query using this recommendation.

The forced primary recommendation is used in two ways by the optimizer:

- The forced primary table is placed first in the join order by the join order optimization strategy.
- The forced primary table is used by the strategy for LPG optimization. The preceding example is a star join since table T1 is joined to the other tables in the query. `t1.c3` is the column used to join T1 to T4. If an index exists over this join column, then it might be advantageous to do preselection against table T1 using the records selected from table T4. The forced primary table recommendation is used as a hint for the optimizer to consider this technique.

Concurrent to run

The preceding logic to identify starvation join can also run in a thread in parallel to the executing query. The AQP handler thread is created for longer running queries. The thread monitors the query execution and can run the same logic described earlier against partial data from the query execution.

If the partial results show starvation join and significant differences with the record count estimates, the query is reoptimized in the thread. When the new plan is ready, the execution of the original plan is stopped and the new plan started. This scheme for correcting join problems 'on the fly' can only be carried out before any records are selected for the final result set.

Note: AQP can help correct query performance problems, but it is not a substitute for a good database design coupled with a good indexing strategy.

Related reference

Adaptive Query Processing in Visual Explain
You can use Visual Explain to request a new plan.

Database Monitor additions for AQP

Additional information is logged in the database monitor when the AQP handler code replaces an executing plan.

A new set of `30xx` records is written to the database monitor reflecting the replaced plan. The user needs to be able to distinguish between records produced for the first plan iteration and records produced for subsequent optimization. To distinguish these records, an unused short integer column of the database monitor record is used as a ‘plan iteration counter'.

Column QQSMINTF is used for this purpose. For the original optimization of the query, the `30xx` records have this field set to *1*. Subsequent reoptimization done by AQP processing will increment the value by 1.

The following is an example of how DB monitor output might look like when *a* is replaced ‘on the fly'. The example query is the following two-file join with an ORDER BY clause over one of the tables:

```
SELECT a.orderkey,b.orderkey
FROM rvdstar/item_fact3 a, rvdstar/item_fact b
WHERE a.quarter - 8 = b.quarter
ORDER BY b.orderkey
```

Assume that an *order by pushdown* plan is chosen, then replaced using AQP while the query is running. The following is an example of what the DB monitor records might look like. The columns shown for the purposes of explaining the changes are QQRID, QQUCNT, QQSMINTF, and QQRCOD. The other fields in the monitor are not affected by AQP processing.

Table 29. Database monitor records for example query

QQRID	QQUCNT	QQSMINTF	QQRCOD
3010	14	-	-
3006	14	1	A0
3001	14	1	I2
3000	14	1	T1
3023	14	1	-
3007	14	1	-

Table 29. Database monitor records for example query (continued)			
QQRID	**QQUCNT**	**QQSMINTF**	**QQRCOD**
3020	14	1	I1
3014	14	1	-
5005	14	1	-
5002	14	1	-
5004	14	1	-
5007	14	1	-
3006	14	2	B6
3000	14	2	T1
3000	14	2	T3
3023	14	2	-
3003	14	2	F7
3007	14	2	-
3020	14	2	I1
3014	14	2	-
5005	14	2	-
5002	14	2	-
5004	14	2	-
1000	14	2	-
5007	14	2	-
3019	14	-	-
1000	14	-	-

Notes on the preceding table:

- There is a full set of optimizer-generated records that reflect the first choice of the optimizer: an *order by pushdown* plan. These records have the QQSMINTF column value set to *1*. There is a 3001 record indicating an index was used to provide the ordering. There are 3000 and 3023 records indicating a Table Scan of the second table and a temporary hash table built to aid join performance. The remaining records, including the 3014 and the 500x records, have QQSMINTF set to *1* to reflect their association with the original *order by pushdown* plan.
- There is a second full set of optimizer-generated records that reflect the second choice of the optimizer: a *sorted temporary* plan to implement the ORDER BY. These records have the QQSMINTF column value set to *2*. This time there are two 3000 records indicating table scan was used to access both tables. There is a 3023 record indicating a temporary hash table was built and a 3003 record indicating the results were sorted. The remaining records, including the 3014 and the 500x records, have QQSMINTF set to *2* to reflect their association with the replacement plan.
- Both sets of optimizer records have the same unique count (QQUCNT value).
- There is a 3006 (Access Plan Rebuilt) record generated for each replacement plan (QQSMINTF > 0). The QQRCOD (reason code) value is set to a new value, *'B6'*. The *'B6'* value indicates the access plan was rebuilt due to AQP processing. In the example, there is a 3006 record with QQSMINTF = *1* and a QQRCOD value of *'A0'*. The *1* indicates that the original optimization built the plan for the first time.

There might not be a 3006 record associated with the original optimization if the optimizer was able to reuse a plan from the plan cache.

- The 1000, 3010 and 3019 records are produced by XPF at open or close time. These records are not generated by the optimizer so there are no changes due to AQP. There are one set of the records, as in previous releases, regardless of whether AQP replaced the plan. The QQSMINTF value is *NULL* for these records.
- The replacement plan is the plan that runs to completion and returns the results. To retrieve the DB monitor records from the plan that actually returns the records, it is necessary to query the DB monitor file using a subquery. Retrieve the records where the QQSMINTF value is equal to the maximum QQSMINTF value for a given QQUCNT.

Related concepts

Database monitor formats
This section contains the formats used to create the database monitor SQL tables and views.

Related reference

Monitoring your queries using the Database Monitor
Start Database Monitor (STRDBMON) command gathers information about a query in real time and stores this information in an output table. This information can help you determine whether your system and your queries are performing well, or whether they need fine-tuning. Database monitors can generate significant CPU and disk storage overhead when in use.

Adaptive Query Processing in Visual Explain
You can use Visual Explain to request a new plan.

QAQQINI query options
There are different options available for parameters in the QAQQINI file.

Row and column access control (RCAC)

Db2 for i introduces row and column access control (RCAC) as an additional layer of data security. RCAC controls access to a table at the row level, column level, or both. RCAC can be used to complement the existing table privileges model.

Indexing Strategy and RCAC

This section focuses on the consequence of RCAC to your SQL query performance when indexing is used.

Row and column access control (RCAC) places access control at the table level around the data itself. SQL rules, which are known as row permissions or column masks, created on rows and columns are the basis of the implementation of this capability.

You can use row and column access control to ensure that your users have access to only the data that is required for their work. For example, tellers in a bank can access customer rows in the CUSTOMER table only from their own branch. All tellers are members of the group user profile TELLER. Customer service representative or telemarketers are members of other groups and allowed to see all rows. A row permission is created by a user who is authorized to the QIBM_DB_SECADM function usage ID.

These SQL rules add additional predicates to any queries or data access requests over tables with defined and activated RCAC permissions. In this example, SQL rules are added to queries over the CUSTOMER table to enforce the following access rules. Depending on the nature of the rules, additional indexes might be advised or existing indexes might need to be enhanced or altered to accommodate the additional predicates enforcing the access. For example, when the TELLER_ROW_ACCESS permission is enabled, additional index advise might include the BRANCH_INFO table and key EMP_ID. In this particular example, index only access can be facilitated by creating an index over BRANCH_INFO that includes EMP_ID and HOME_BRANCH as key fields. The first to facilitate the probe, the second to prevent unnecessary access to the BRANCH_INFO table.

```
CREATE PERMISSION TELLER_ROW_ACCESS ON CUSTOMER
-------------------------------------------------------
-- Teller information:
-- Group TELLER is allowed to access customer data only
-- in their branch.
------------------------------------------------------------
```

```
FOR ROWS WHERE VERIFY_GROUP_FOR_USER(SESSION_USER, 'TELLER') = 1
AND
BRANCH = (SELECT HOME_BRANCH FROM BRANCH_INFO WHERE EMP_ID = SESSION_USER)
ENFORCED FOR ALL ACCESS
ENABLE;
```

```
ALTER TABLE CUSTOMER ACTIVATE ROW ACCESS CONTROL;
```

In the example below, not only are you verifying certain user groups for access to particular patient records but also masking certain data based on whether the patient has participated in a clinical trial. Extra security is that physicians can see only patient records for whom they are the primary care provider.

```
CREATE PERMISSION PCP ON patient
-------------------------------------------------------
-- Primary Care Physician Access
-- Group PCP is allowed to access patient data only
-- AND the Primary Care Physician must be assigned to patient
-- Group RESEARCH are allowed to access patient data for those patients
-- that opted in to a clinical trial
------------------------------------------------------------
FOR ROWS WHERE
(VERIFY_GROUP_FOR_USER(SESSION_USER, 'PCP') = 1
AND
PCPID = (SELECT PCPID FROM PHYSICIAN WHERE PCPUSER = SESSION_USER) )
OR
(VERIFY_GROUP_FOR_USER(SESSION_USER, 'RESEARCH') = 1
AND
(SELECT 1 FROM PATIENTCHOICE C
WHERE PATIENT.patientid = C.patientid
AND C.CHOICE = 'clinical trial'
AND C.VALUE = 'opt-in')=1
)
ENFORCED FOR ALL ACCESS
ENABLE;
```

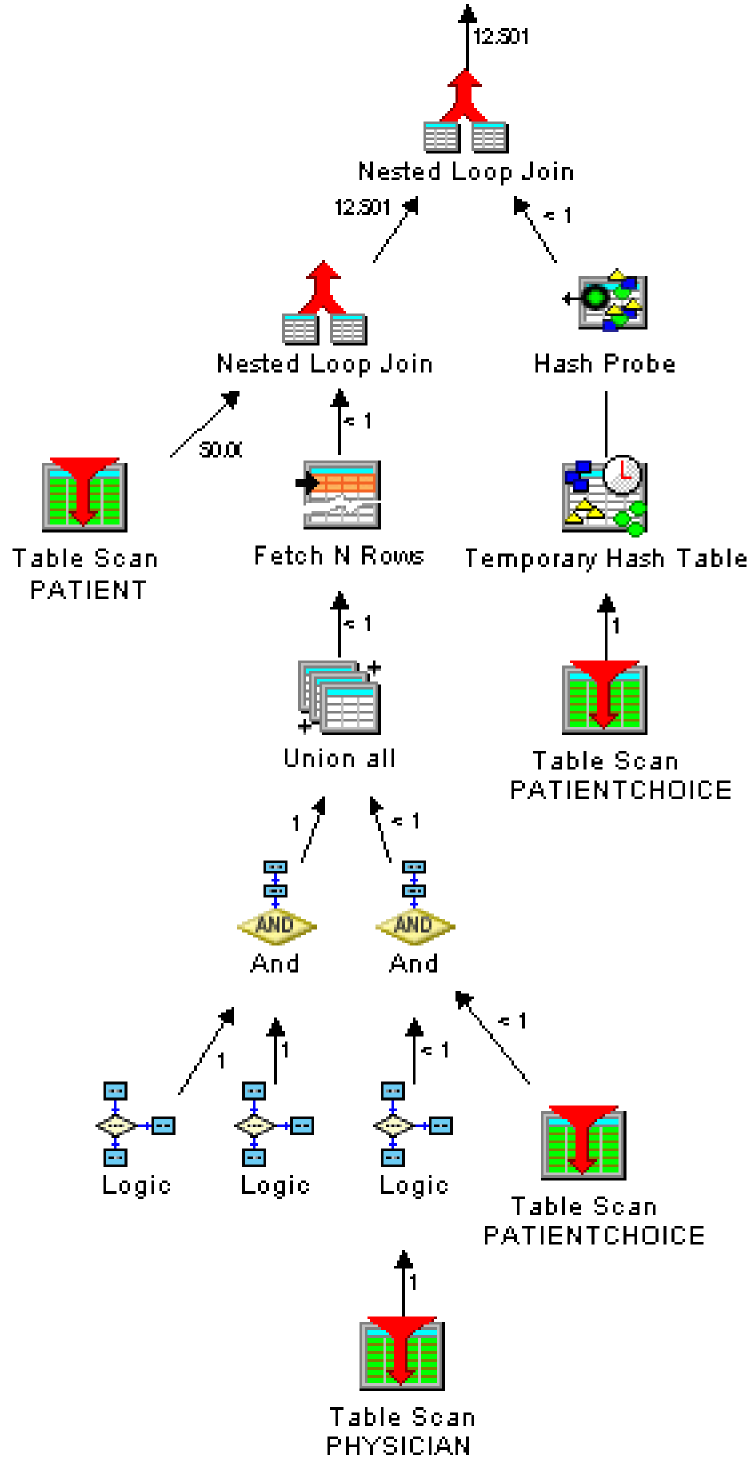
```
CREATE MASK PHARMACY_MASK ON PATIENT FOR
--------------------------------------------------------
-- Medical information:
-- Group PCP is allowed to access the full information in column PHARMACY.
-- For the purposes of drug research, Role DRUG_RESEARCH can
-- conditionally see a patient's medical information
-- provided that the patient has opted-in.
-- In all other cases, null values are rendered as column
-- values.
----------------------------------------------------
COLUMN PHARMACY RETURN
         CASE WHEN
          VERIFY_GROUP_FOR_USER(SESSION_USER,'PCP') = 1 OR
          (VERIFY_GROUP_FOR_USER(SESSION_USER,'DRUG_RSRCH')=1
                    AND
                    (SELECT 1 FROM PATIENTCHOICE C
                   WHERE PATIENT.patientid = C.patientid
                           AND C.CHOICE = 'drug-research'
                           AND C.VALUE = 'opt-in')= 1
                  )
         THEN PHARMACY
         ELSE NULL
END
ENABLE;
```

```
ALTER TABLE PATIENT ACTIVATE ROW ACCESS CONTROL ACTIVATE COLUMN ACCESS CONTROL;
```

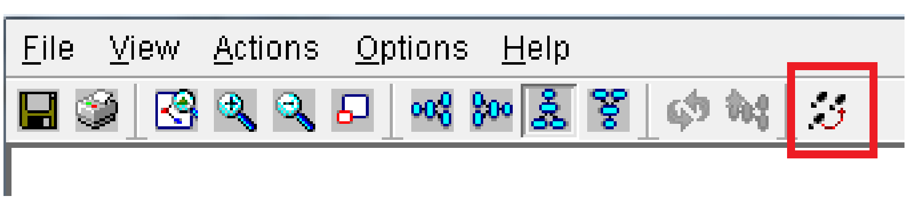
The query in the next example, before the introduction of RCAC policies would have accessed only the PATIENT table. Now it accesses the PATIENT table and the supporting tables that are associated with the row and column permissions.

The next graphic is the Visual Explain for the next example query. As you can see, the PATIENT table is accessed along with any other tables mentioned in the ROW and COLUMN access control.

SELECT * FROM PATIENT WHERE PATIENTID = ?

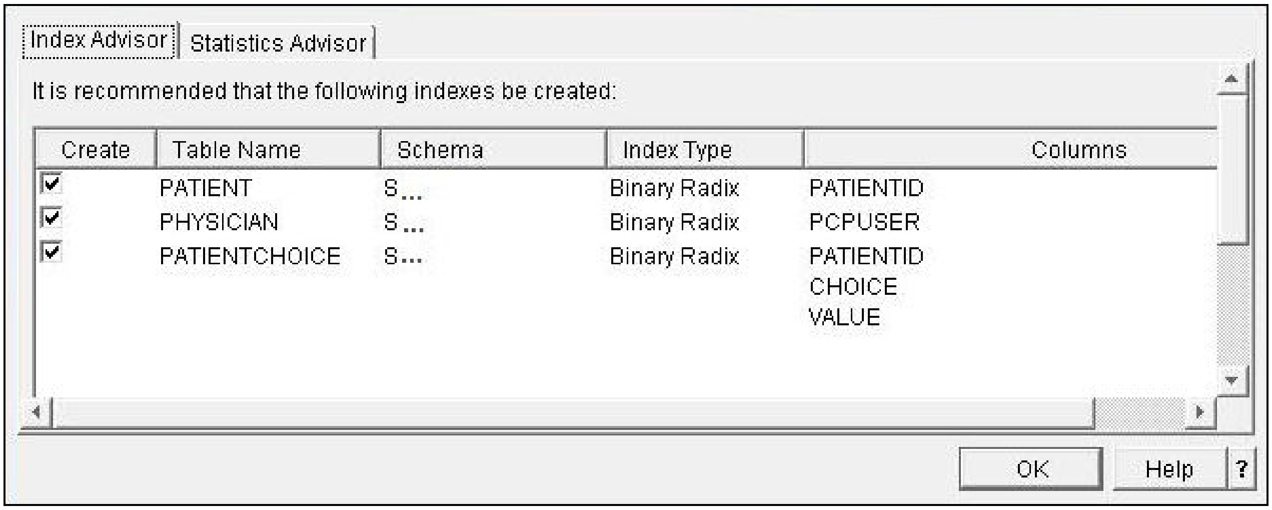


By clicking the index advised icon that is shown in the next graphic:



You get the resulting index advice depicted in the next graphic that shows that it is not only over the PATIENT table that is explicitly specified in the query, but also over the supporting RCAC tables.

Not considering additional advice per the introduction of RCAC SQL rules can affect query performance.



Materialized query tables and RCAC

This section focuses on the consequence of RCAC to your SQL query performance when MQTs are used.

Materialized Query Tables (MQTs) are heavily relied upon by data warehousing applications for better query performance. RCAC and MQTs coexist in harmony. This means:

1. MQTs must continue to provide their added performance benefit to data warehousing applications.
2. MQTs cannot become a means for gaining access to data protected through RCAC rules that are specified in the dependent base tables, either through direct access to the MQT or by MQT matching and substitution.

If a materialized query table that depends on the table (directly or indirectly through a view) for which access control is being activated and that materialized query table does not already have its own access control activated, row level access control is implicitly activated for the materialized query table. This restricts direct access to the contents of the materialized query table. A query that explicitly references the MQT table before such a row permission is defined returns Row Not Found as if there was no data in the table.

In this example MQT:

```
CREATE TABLE MQT1
AS (SELECT patientid, patientname,pcpid,pharmacy
FROM  patient
WHERE diagnosis is not null)
DATA INITIALLY IMMEDIATE REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER;
```

To provide access to this materialized query table, an appropriate row permission can be created, or an ALTER TABLE DEACTIVATE ROW ACCESS CONTROL on the materialized query table can be issued to remove the row level protection if that is appropriate. If the query optimizer substitutes one or more tables in a query with this materialized query table via MQT substitution, the row and column access controls on the replaced (base) tables remain in effect, and the access controls, if any, on the materialized query table do not apply.

```
SELECT * FROM MQT1
```

results in no rows because it does not have its own RCAC policy and therefore it cannot expose rows per the PATIENT table.

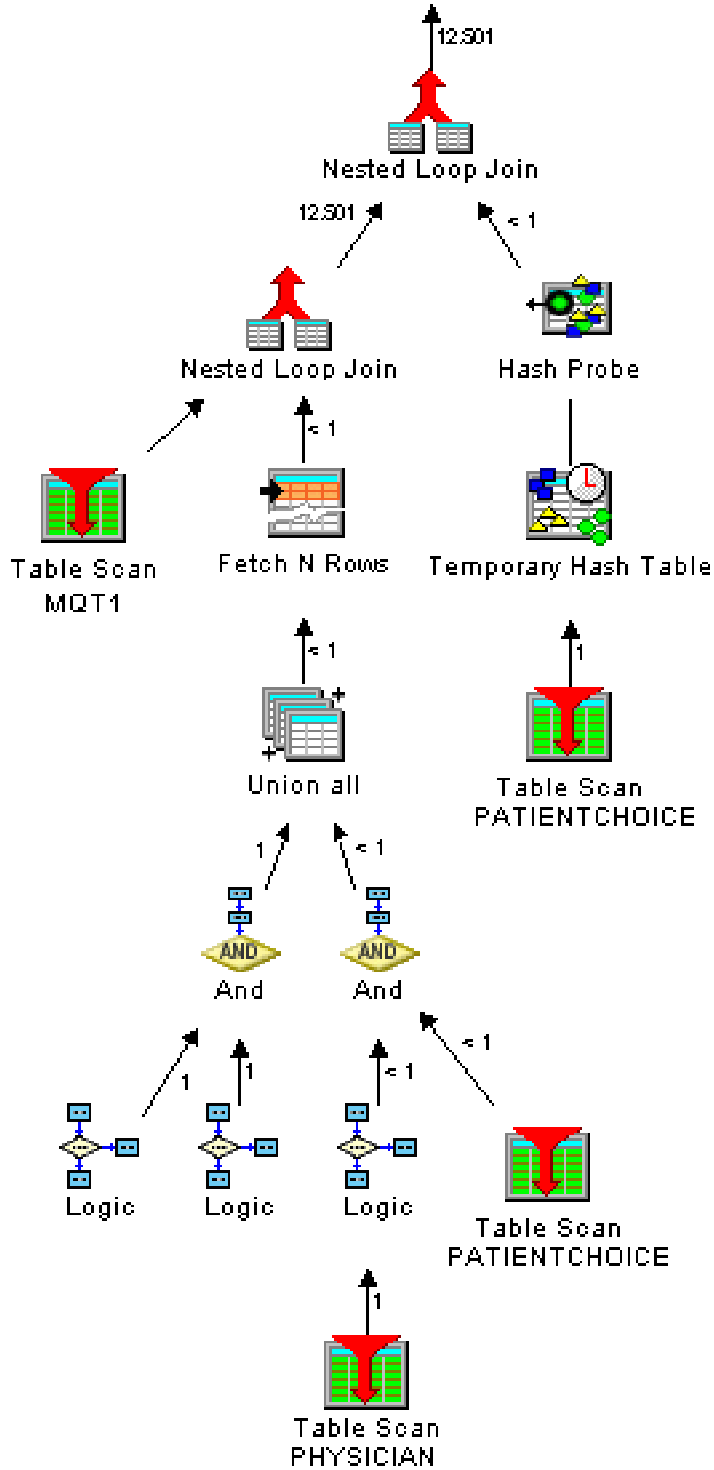
The following query however can be satisfied by the MQT1

```
SELECT patientid, patientname, pharmacy  FROM patient  WHERE  patientid>4 and diagnosis is not
null;
```

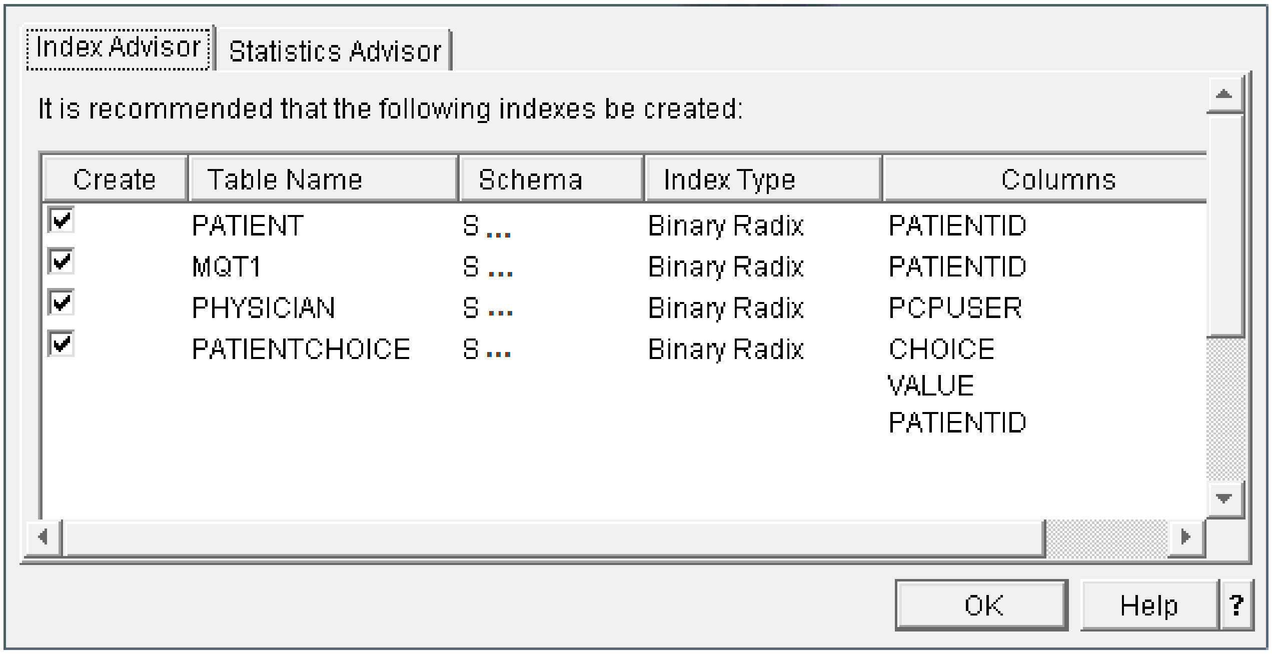
Row and column level access control does not affect the REFRESH TABLE statement. The table is refreshed as if row and column level access controls do not exist.

```
REFRESH  TABLE  mqt1;
```

The graphic below shows the Visual Explain that reflects the MQT match and substitution. Note that had the MQT1 not surfaced a required value of PCPID for the existing RCAC SQL Rules, it would not be able to satisfy the query request as an MQT match, even though that field is not in the required select list. In this example Visual Explain, you can see the MQT1 substituted but also inherited the RCAC rules of the base table PATIENT.



Index advice of the originating query, which is depicted in the graphic below, includes advice over the main query table, over the MQT and over the RCAC required tables.



As many MQTs, such as the one below, provide ready made aggregation values so aggregating queries in a data warehousing environment perform quickly, these MQTs are now likely not to match query requests with aggregated selection via MQT substitution.

The aggregation is based on the REFRESH TABLE with no RCAC applied and yet the matching is based on the underlying base table and all its RCAC requirements.

```
CREATE TABLE MQT_AGG
AS (SELECT pcpid, count(*) patientcnt
FROM  patient group by pcpid
)
DATA INITIALLY IMMEDIATE REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY USER;
```

The following query, although it appears to be a match for the above MQT_AGG, will not substitute the MQT per RCAC rules.

```
SELECT pcpid, count(*)  FROM PATIENT WHERE pcpid in ( 1, ...) GROUP BY pcpid
```

All existing MQTs should be analyzed before deploying RCAC policy on base tables to make sure that performance does not unexpectedly start to suffer because MQTs are no longer available to facilitate the request.

Because most aggregating queries are not dealing with 'details' and so possibly less sensitive to the requirements of RCAC, aggregating MQT over base tables with RCAC might be best deployed by direct substitution in the query and restriction through table privileges and disabling the default RCAC rule, restricting all rows, as follows.

```
ALTER TABLE MQT_AGG DEACTIVATE ROW ACCESS CONTROL;
```

This deactivates the default RCAC applied due to base tables with RCAC and allows direct access to the MQT in a warehousing environment.

Optimizing query performance using query optimization tools

Query optimization is an iterative process. You can gather performance information about your queries and control the processing of your queries.

DB2 for IBM i – Health Center

Use the DB2 for IBM i Health Center to capture information about your database. You can view the total number of objects, the size limits of selected objects, the design limits of selected objects, environmental limits, and activity level.

Navigator view of Health Center

The System i Navigator provides a robust graphical interface to capture, view, and interact with the Health Center.

To start the health center, follow these steps:

1. In the System i Navigator window, expand the system that you want to use.
2. Expand **Databases**.
3. Right-click the database that you want to work with and select **Health Center**.

You can change your preferences by clicking **Change** and entering filter information. Click **Refresh** to update the information.

To save your health center history, do the following:

1. In the System i Navigator window, expand the system you want to use.
2. Expand **Databases**.
3. Right-click the database that you want to work with and select **Health Center**.
4. On the health center dialog, select the area that you want to save. For example, if you want to save the current overview, click **Save** on the Overview tab. Size limits and Design limits are not saved.
5. Specify a schema and table to save the information. You can view the contents of the selected table by clicking **View Contents**. If you select to save information to a table that does not exist, the system creates the table for you.

Health Center SQL procedures

The Health Center is implemented upon several DB2 for i SQL procedures.

IBM i users can call the Health Center SQL procedures directly.

QSYS2.Health_Database_Overview ()

The QSYS2.Health_Database_Overview() procedure returns counts of all the different types of DB2 for i objects within the target schema or schemas. The counts are broken down by object type and subtype.

Procedure definition:

```
CREATE PROCEDURE QSYS2.HEALTH_DATABASE_OVERVIEW(
       IN ARCHIVE_OPTION INTEGER,
       IN OBJECT_SCHEMA VARCHAR(258),
       IN NUMBER_OF_ITEMS_ARCHIVE INTEGER,
       IN OVERVIEW_SCHEMA VARCHAR(258),
       IN OVERVIEW_TABLE VARCHAR(258))
       DYNAMIC RESULT SETS 1
       LANGUAGE C
       SPECIFIC QSYS2.HEALTH_DATABASE_OVERVIEW
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'QSYS/QSQHEALTH(OVERVIEW)'
       PARAMETER STYLE SQL;
```

Service Program Name: QSYS/QSQHEALTH

Default Public Authority: *USE

Threadsafe: Yes

IBM i release

This procedure was added to IBM i in V5R4M0.

Parameters

Archive_Option
(Input) The type of operation to perform for the DB2 for i Health Center overview detail.

The supported values are:

- 1 = Query only, no archive action is taken
- 2 = Archive only
- 3 = Create archive and archive
- 4 = Query the archive

Note: Option 1 produces a new result set. Options 2 and 3 simply use the results from the last Query option. Option 3 fails if the archive exists.

Object_Schema
(Input) The target schema or schemas for this operation. A single schema name can be entered. The '%' character can be used to direct the procedure to process all schemas with names that start with the same characters which appear before the '%'. When this parameter contains only the '%' character, the procedure processes all schemas within the database.

Number_Of_Items_Archive
(Input) The number of rows to archive.

The archive can be used to recognize trends over time. To have meaningful historical comparisons, choose the row count size carefully. This argument is ignored if the Archive_Option is 1.

Overview_Table
(Input) The table that contains the database overview archive.

This argument is ignored if the Archive_Option is 1.

Authorities

To query an existing archive, *USE object authority is required for the Overview_Schema and Overview_Table. To create an archive, *CHANGE object authority is required for the Overview_Schema. To add to an existing archive, *CHANGE object authority is required for the Overview_Table and *USE object authority is required for the Overview_Schema.

Result Set

When Archive_Option is 1 or 4, a single result set is returned.

The format of the result is as follows.

QSYS2.Health_Database_Overview () result set format:

```
"TIMESTAMP" TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
SCHEMAS BIGINT NOT NULL ,
GRP01 CHAR(1) DEFAULT NULL ,
TABLES BIGINT NOT NULL ,
PARTITIONED_TABLES FOR COLUMN TABLESRT BIGINT NOT NULL ,
DISTRIBUTED_TABLES FOR COLUMN TABLES_DST BIGINT NOT NULL ,
MATERIALIZED_QUERY_TABLES FOR COLUMN TABLES_MAT BIGINT NOT NULL ,
PHYSICAL_FILES FOR COLUMN TABLESHY BIGINT NOT NULL ,
SOURCE_FILES FOR COLUMN TABLES_SRC BIGINT NOT NULL ,
GRP02 CHAR(1) DEFAULT NULL ,
VIEWS BIGINT NOT NULL ,
LOGICAL_FILES FOR COLUMN VIEWS_LGL  BIGINT NOT NULL ,
```

```
GRP03 CHAR(1) DEFAULT NULL ,
BINARY_RADIX_INDEXES FOR COLUMN INDEXES_BI BIGINT NOT NULL ,
EVI_INDEXES FOR COLUMN INDEXES_EV BIGINT NOT NULL ,
GRP04 CHAR(1) DEFAULT NULL ,
PRIMARY_KEY_CONSTRAINTS FOR COLUMN CSTSRI   BIGINT NOT NULL ,
UNIQUE_CONSTRAINTS FOR COLUMN CSTS_UNQ   BIGINT NOT NULL ,
CHECK_CONSTRAINTS FOR COLUMN CSTS_CHK   BIGINT NOT NULL ,
REFERENTIAL_CONSTRAINTS FOR COLUMN CSTS_RI    BIGINT NOT NULL ,
GRP05 CHAR(1) DEFAULT NULL ,
EXTERNAL_TRIGGERS FOR COLUMN TRGS_EXT   BIGINT NOT NULL ,
SQL_TRIGGERS FOR COLUMN TRGS_SQL   BIGINT NOT NULL ,
INSTEAD_OF_TRIGGERS FOR COLUMN TRGS_INSTD BIGINT NOT NULL ,
GRP06 CHAR(1) DEFAULT NULL ,
ALIASES BIGINT NOT NULL ,
DDM_FILES BIGINT NOT NULL ,
GRP07 CHAR(1) DEFAULT NULL ,
EXTERNALROCEDURES FOR COLUMN PROCS_EXT  BIGINT NOT NULL ,
SQLROCEDURES FOR COLUMN PROCS_SQL  BIGINT NOT NULL ,
GRP08 CHAR(1) DEFAULT NULL ,
EXTERNAL_SCALAR_FUNCTIONS FOR COLUMN FUNCS_EXTS BIGINT NOT NULL ,
EXTERNAL_TABLE_FUNCTIONS FOR COLUMN FUNCS_EXTT BIGINT NOT NULL ,
SOURCE_SCALAR_FUNCTIONS FOR COLUMN FUNCS_SRCS BIGINT NOT NULL ,
SOURCE_AGGREGATE_FUNCTIONS FOR COLUMN FUNCS_SRCA BIGINT NOT NULL ,
SQL_SCALAR_FUNCTIONS FOR COLUMN FUNCS_SQLS BIGINT NOT NULL ,
SQL_TABLE_FUNCTIONS FOR COLUMN FUNCS_SQLT BIGINT NOT NULL ,
GRP09 CHAR(1) DEFAULT NULL ,
SEQUENCES BIGINT NOT NULL ,
SQLACKAGES FOR COLUMN SQLPKGS    BIGINT NOT NULL ,
USER_DEFINED_DISTINCT_TYPES FOR COLUMN UDTS BIGINT NOT NULL ,
JOURNALS BIGINT NOT NULL ,
JOURNAL_RECEIVERS FOR COLUMN JRNRCV BIGINT NOT NULL ,
"SCHEMA" VARCHAR(258) ALLOCATE(10) NOT NULL
```

```
LABEL ON COLUMN <result set>
( "TIMESTAMP" IS 'Timestamp' ,
    SCHEMAS IS 'Schemas' ,
    GRP01 IS 'Tables' ,
    TABLES IS 'Non-partitioned     tables' ,
    PARTITIONED_TABLES IS 'Partitioned         tables' ,
    DISTRIBUTED_TABLES IS 'Distributed         tables' ,
    MATERIALIZED_QUERY_TABLES IS 'Materialized        query               tables' ,
    PHYSICAL_FILES IS 'Physical            files' ,
    SOURCE_FILES IS 'Source              files' ,
    GRP02 IS 'Views' ,
    VIEWS IS 'Views' ,
    LOGICAL_FILES IS 'Logical             files' ,
    GRP03 IS 'Indexes' ,
    BINARY_RADIX_INDEXES IS 'Binary              radix               indexes' ,
    EVI_INDEXES IS 'Encoded             vector              indexes' ,
    GRP04 IS 'Constraints' ,
    PRIMARY_KEY_CONSTRAINTS IS 'PRIMARY KEY         constraints' ,
    UNIQUE_CONSTRAINTS IS 'UNIQUE              constraints' ,
    CHECK_CONSTRAINTS IS 'CHECK               constraints' ,
    REFERENTIAL_CONSTRAINTS IS 'Referential         constraints' ,
    GRP05 IS 'Triggers' ,
    EXTERNAL_TRIGGERS IS 'External            triggers' ,
    SQL_TRIGGERS IS 'SQL                 triggers' ,
    INSTEAD_OF_TRIGGERS IS 'INSTEAD OF          triggers' ,
    GRP06 IS 'Aliases' ,
    ALIASES IS 'Aliases' ,
    DDM_FILES IS 'DDM                 files' ,
    GRP07 IS 'Procedures' ,
    EXTERNALROCEDURES IS 'External            procedures' ,
    SQLROCEDURES IS 'SQL                 procedures' ,
    GRP08 IS 'Functions' ,
    EXTERNAL_SCALAR_FUNCTIONS IS 'External            scalar              functions' ,
    EXTERNAL_TABLE_FUNCTIONS IS 'External            table               functions' ,
    SOURCE_SCALAR_FUNCTIONS IS 'Source              scalar              functions' ,
    SOURCE_AGGREGATE_FUNCTIONS IS 'Source              aggregate           functions' ,
    SQL_SCALAR_FUNCTIONS IS 'SQL                 scalar              functions' ,
    SQL_TABLE_FUNCTIONS IS 'SQL                 table               functions' ,
    GRP09 IS 'Miscellaneous' ,
    SEQUENCES IS 'Sequences' ,
    SQLACKAGES IS 'SQL                 packages' ,
    USER_DEFINED_DISTINCT_TYPES IS 'User-defined        distinct            types' ,
    JOURNALS IS 'Journals' ,
    JOURNAL_RECEIVERS IS 'Journal             receivers' ,
    "SCHEMA" IS 'Schema              mask' ) ;
```

Error Messages

Table 30. Error messages	
Message ID	**Error Message Text**
SQL0462 W	This warning appears in the job log if the procedure encounters objects for which the user does not have *USE object authority. The warning is provided as an indication that the procedure was unable to process all available objects.

Usage Notes

None

Related Information

None

Examples

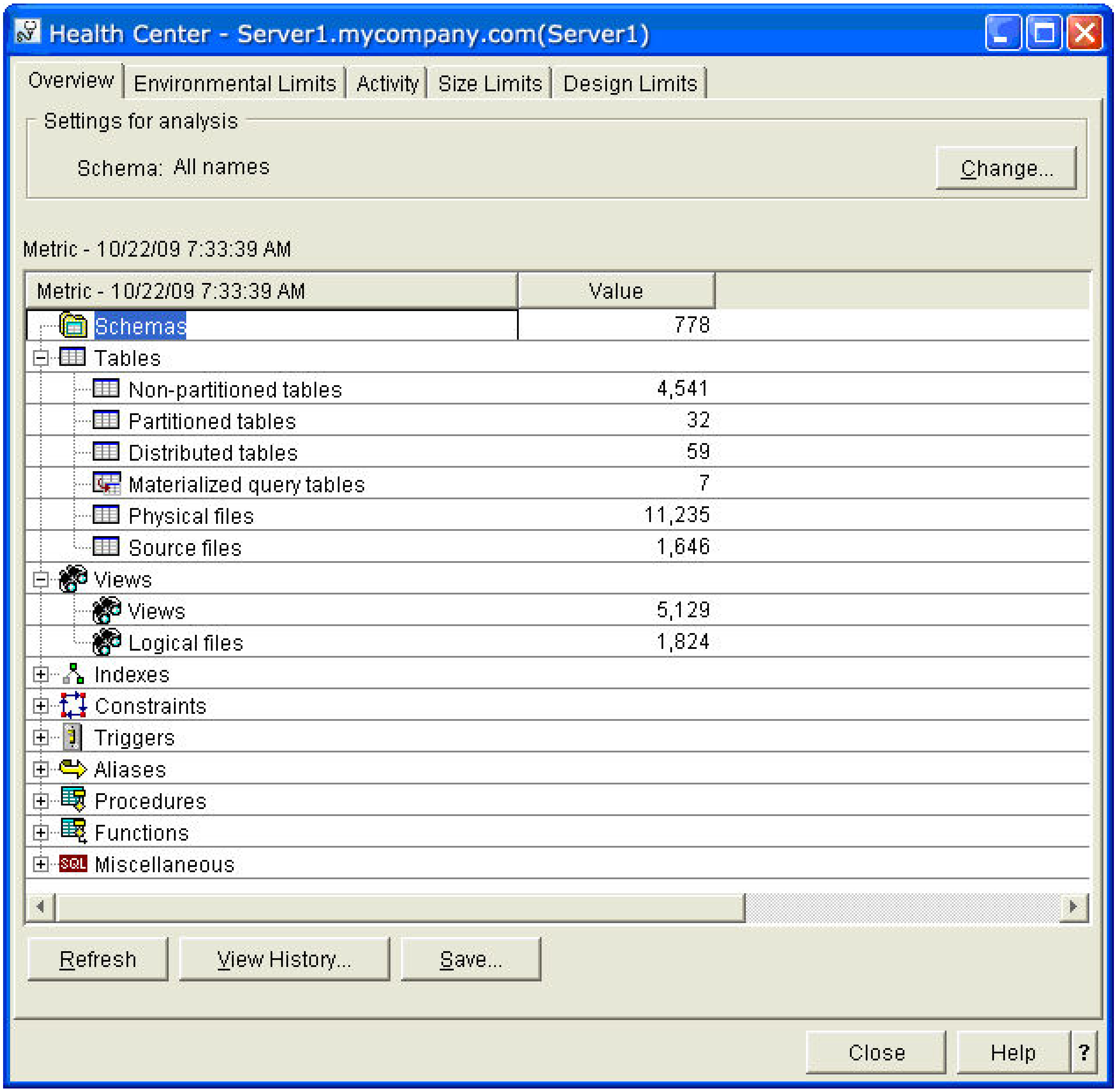
Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 645.

Example 1

Retrieve the overview for the entire database.

```
CALL QSYS2.Health_Database_Overview(1, '%', NULL, NULL, NULL);
```

Example results in System i Navigator:



Example 2

Archive all rows in the overview to an SQL table named MYLIB/ARCHIVE1.

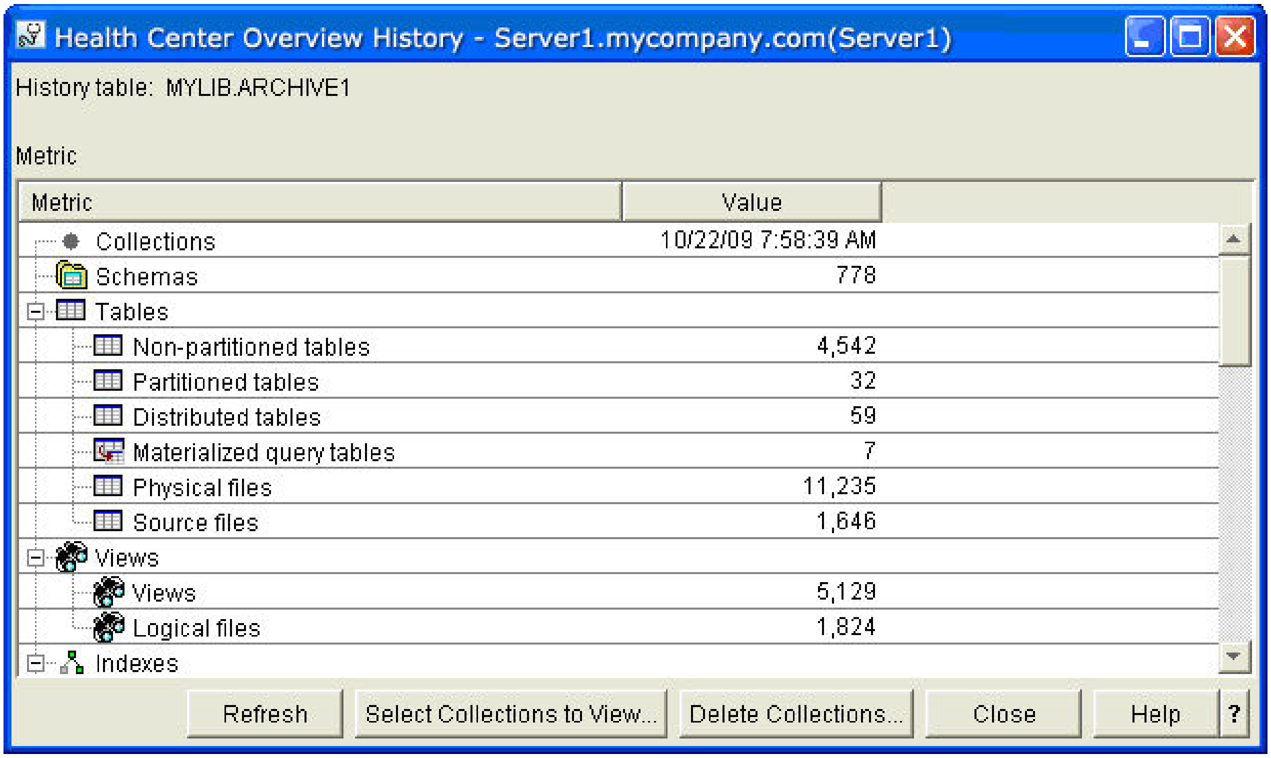
```
CALL QSYS2.Health_Database_Overview(3, '%', 2147483647, 'MYLIB', 'ARCHIVE1')
```

Example 3

Retrieve the overview from MYLIB/ARCHIVE1.

```
CALL QSYS2.Health_Database_Overview(4, '%', NULL, 'MYLIB', 'ARCHIVE1')
```

Example results in System i Navigator:



QSYS2.Health_Activity ()
The QSYS2.Health _Activity () procedure returns summary counts of database and SQL operations over a set of objects within one or more schemas.

Procedure definition:

```
CREATE PROCEDURE QSYS2.HEALTH_ACTIVITY(
       IN ARCHIVE_OPTION INTEGER,
       IN REFRESH_CURRENT_VALUES INTEGER,
       IN OBJECT_SCHEMA VARCHAR(258),
       IN OBJECT_NAME VARCHAR(258),
       IN NUMBER_OBJECTS_ACTIVITY_TO_ARCHIVE INTEGER,
       IN NUMBER_OF_ACTIVITY_ARCHIVE INTEGER,
       IN ACTIVITY_SCHEMA VARCHAR(258),
       IN ACTIVITY_TABLE VARCHAR(258))
       DYNAMIC RESULT SETS 1
       LANGUAGE C
       SPECIFIC QSYS2.HEALTH_ACTIVITY
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'QSYS/QSQHEALTH(ACTIVITY)'
       PARAMETER STYLE SQL;
```

Service Program Name: QSYS/QSQHEALTH

Default Public Authority: *USE

Threadsafe: Yes

IBM i release

This procedure was added to IBM i 6.1.

Parameters

Archive_Option
(Input) The type of operation to perform for the DB2 for i Health Center overview detail.

The supported values are:

- 1 = Query only, no archive action is taken
- 2 = Archive only
- 3 = Create archive and archive
- 4 = Query the archive

Note: Option 1 produces a new result set. Options 2 and 3 simply use the results from the last Query option. Option 3 fails if the archive exists.

Refresh_Current_Values
(Input) This option directs how the archive operation is done. This option is only valid with archive options 2 and 3.

The supported values are:

- 0 = No. Indicates that we capture the activity on the entire set of specified schemas and objects.
- 1 = Yes. Indicates that we only refresh the activity of the objects previously captured (based on the short names).
- 2 = None. Use the results from the prior call. A call must have been performed in this job before using this option

Object_Schema
(Input) The target schema or schemas for this operation. A single schema name can be entered. The '%' character can be used to direct the procedure to process all schemas with names that start with the same characters which appear before the '%'. When this parameter contains only the '%' character, the procedure processes all schemas within the database.

This name also affects the items refreshed if Refresh_Current_Values = 1.

Object_Name
(Input) The target object name for this operation. Only the '%' character is treated as a wildcard since an underscore is a valid character in a name. The name must be delimited, if necessary, and case sensitive.

This name also affects the items refreshed if Refresh_Current_Values = 1.

Number_Objects_Activity_to_Archive
(Input) The number of objects to save for each activity.

Number_Of_Activity_Archive
(Input) The number of rows to save per object activity.

The archive can be used to recognize trends over time. To have meaningful historical comparisons, choose the row count size carefully. This argument is ignored if the Archive_Option is 1 or 4.

Activity_Schema
(Input) The table that contains the database activity archive.

This argument is ignored if the Archive_Option is 1.

Activity_Table
The table that contains the database activity archive.

This argument is ignored if the Archive_Option is 1.

Authorities

To query an existing archive, *USE object authority is required for the Activity_Schema and Activity_Table. To create an archive, *CHANGE object authority is required for the Activity_Schema. To add to an existing archive, *CHANGE object authority is required for the Activity_Table and *USE object authority is required for the Activity_Schema.

When Archive_Option is 1 or 3, *USE object authority is required for the Object_Schema and for any objects which are indicated by Object_Name. When an object is encountered and the caller does not have

*USE object authority, an SQL0462 warning is placed in the job log. The object is skipped and not included in the procedure result set.

Result Set

When Archive_Option is 1 or 4, a single result set is returned.

The format of the result is as follows. All these items were added for IBM i 6.1.

QSYS2.Health_Activity() result set format:

```
"TIMESTAMP" TIMESTAMP NOT NULL,
ACTIVITY VARCHAR(2000) ALLOCATE(20) DEFAULT NULL,
CURRENT_VALUE FOR COLUMN "VALUE" BIGINT DEFAULT NULL,
OBJECT_SCHEMA FOR COLUMN BSCHEMA VARCHAR(128)ALLOCATE(10) DEFAULT NULL,
OBJECT_NAME FOR COLUMN BNAME VARCHAR(128) ALLOCATE(20) DEFAULT NULL,
OBJECT_TYPE FOR COLUMN BTYPE VARCHAR(24) ALLOCATE(10) DEFAULT NULL,
SYSTEM_OBJECT_SCHEMA FOR COLUMN SYS_DNAME VARCHAR(10) ALLOCATE(10)DEFAULT NULL,
SYSTEM_OBJECT_NAME FOR COLUMN SYS_ONAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
PARTITION_NAME FOR COLUMN MBRNAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
ACTIVITY_ID FOR COLUMN ACTIV00001 INTEGER DEFAULT NULL
```

```
LABEL ON COLUMN <result set>
("TIMESTAMP" IS 'Timestamp',
    ACTIVITY IS 'Activity',
    CURRENT_VALUE IS 'Current             Value',
    OBJECT_SCHEMA IS 'Object              Schema',
    OBJECT_NAME IS 'Object              Name',
    OBJECT_TYPE IS 'Object              Type',
    SYSTEM_OBJECT_SCHEMA IS 'System              Object              Schema',
    SYSTEM_OBJECT_NAME IS 'System              Object              Name',
    PARTITION_NAME IS 'Partition           Name',
    ACTIVITY_ID IS 'Activity            ID');
```

Limit Detail

The supported Database Health Center Activity can be seen on any machine by executing this query. The supported value column contains zeros because this category of Health Center information is not tied to a limit.

```
SELECT * FROM QSYS2.SQL_SIZING WHERE SIZING_ID BETWEEN 18000 AND 18199;
```

Note: The **bold** rows were added in IBM i 7.1.

Table 31. Summary counts of database and SQL operations within a schema.

SIZING_ID	SIZING_NAME	SUPPORTED_VALUE
18100	INSERT OPERATIONS	0
18101	UPDATE OPERATIONS	0
18102	DELETE OPERATIONS	0
18103	LOGICAL READS	0
18104	PHYSICAL READS	0
18105	CLEAR OPERATIONS	0
18106	INDEX BUILDS/REBUILDS	0
18107	DATA SPACE REORGANIZE OPERATIONS	0
18108	DATA SPACE COPY OPERATIONS	0
18109	FULL OPENS	0
18110	FULL CLOSES	0

Table 31. Summary counts of database and SQL operations within a schema. (continued)		
SIZING_ID	**SIZING_NAME**	**SUPPORTED_VALUE**
18111	DAYS USED	0
18112	INDEX QUERY USE	0
18113	INDEX QUERY STATISTICS USE	0
18114	INDEX LOGICAL READS	0
18115	**INDEX RANDOM READS**	**0**
18116	SQL STATEMENT COMPRESSION COUNT	0
18117	SQL STATEMENT CONTENTION COUNT	0
18118	**RANDOM READS**	**0**
18119	**SEQUENTIAL READS**	**0**

Error Messages

Table 32. Error messages	
Message ID	**Error Message Text**
SQL0462 W	This warning appears in the job log if the procedure encounters objects for which the user does not have *USE object authority. The warning is provided as an indication that the procedure was unable to process all available objects.

Usage Notes

None

Related Information

None

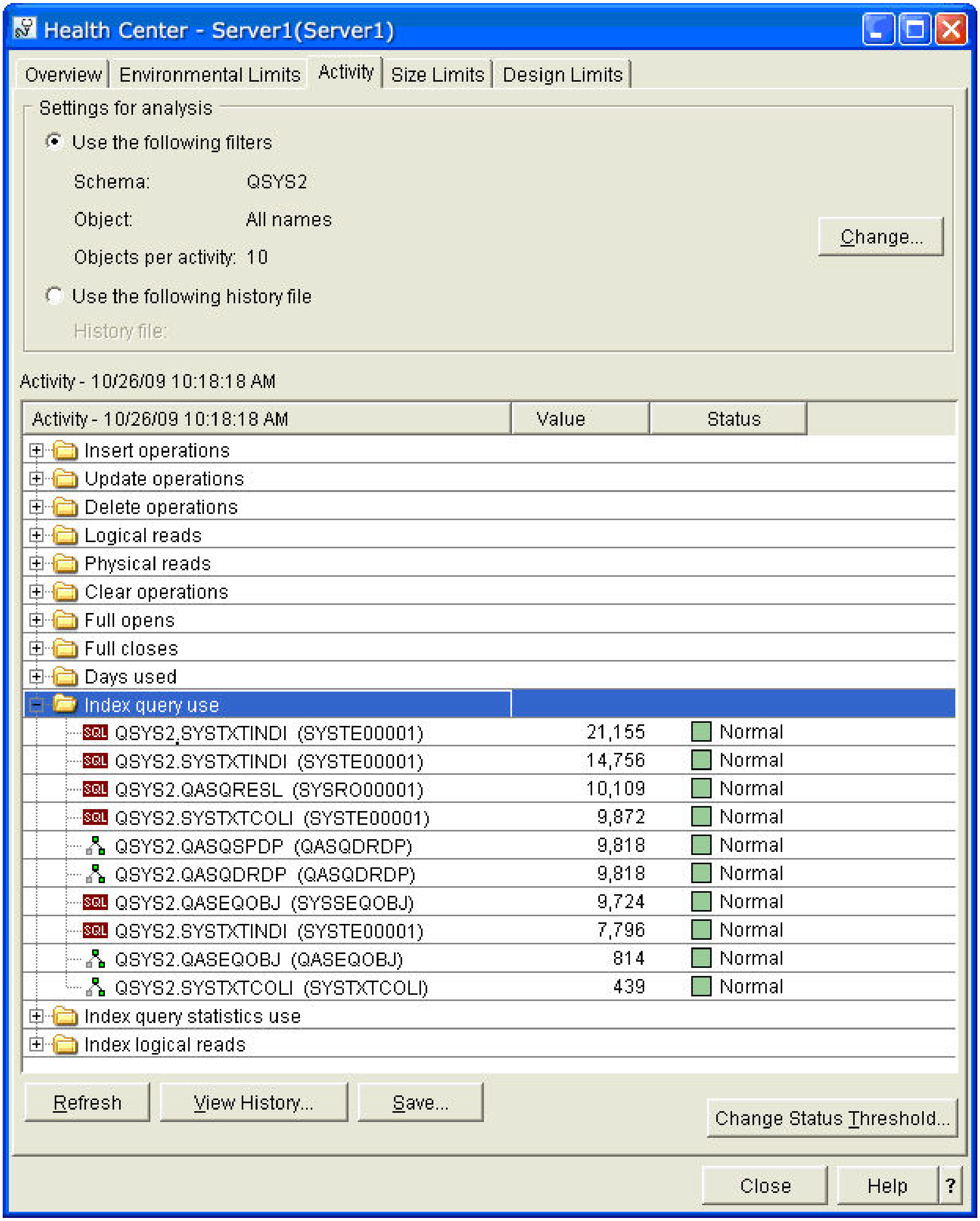
Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 645.

Retrieve the activity information for all objects within the QSYS2 schema, using a maximum of 10 objects per each activity.

```
CALL QSYS2.Health_Activity(1, 0, 'QSYS2', '%', 10, NULL, NULL, NULL);
```

Example results in System i Navigator:


Health Center - Server1(Server1)
Overview
Environmental Limits
Activity
Size Limits
Design Limits
Settings for analysis
Use the following filters
Schema: QSYS2
Object: All names
Objects per activity: 10
Change...
Use the following history file
History file:
Activity - 10/26/09 10:18:18 AM
Activity - 10/26/09 10:18:18 AM
Value
Status
Insert operations
Update operations
Delete operations
Logical reads
Physical reads
Clear operations
Full opens
Full closes
Days used
Index query use
QSYS2.SYSTXTINDI (SYSTE00001) 21,155 Normal
QSYS2.SYSTXTINDI (SYSTE00001) 14,756 Normal
QSYS2.QASQRESL (SYSRO00001) 10,109 Normal
QSYS2.SYSTXTCOLI (SYSTE00001) 9,872 Normal
QSYS2.QASQSPDP (QASQDRDP) 9,818 Normal
QSYS2.QASQDRDP (QASQDRDP) 9,818 Normal
QSYS2.QASEQOBJ (SYSSEQOBJ) 9,724 Normal
QSYS2.SYSTXTINDI (SYSTE00001) 7,796 Normal
QSYS2.QASEQOBJ (QASEQOBJ) 814 Normal
QSYS2.SYSTXTCOLI (SYSTXTCOLI) 439 Normal
Index query statistics use
Index logical reads
Refresh
View History...
Save...
Change Status Threshold...
Close
Help
?

QSYS2.Health_Design_Limits ()

The QSYS2.Health_Design_Limits () procedure returns detailed counts of design limits over a set of objects within one or more schemas. Design limits correspond to architectural constructs, such as ‘Maximum number of columns in a table or view'.

Procedure definition:

```
CREATE PROCEDURE QSYS2.HEALTH_DESIGN_LIMITS(
       ARCHIVE_OPTION INTEGER,
       IN REFRESH_CURRENT_VALUES INTEGER,
       IN OBJECT_SCHEMA VARCHAR(258),
       IN OBJECT_NAME VARCHAR(258),
       IN NUMBER_OBJECTS_LIMIT_TO_ARCHIVE INTEGER,
       IN NUMBER_OF_LIMITS_ARCHIVE INTEGER,
       IN LIMIT_SCHEMA VARCHAR(258),
       IN LIMIT_TABLE VARCHAR(258),
       DYNAMIC RESULT SETS 1
       LANGUAGE C
       SPECIFIC QSYS2.HEALTH_DESIGN_LIMITS
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'QSYS/QSQHEALTH(DESIGN)'
       PARAMETER STYLE SQL;
```

Service Program Name: QSYS/QSQHEALTH

Default Public Authority: *USE

Threadsafe: Yes

IBM i release

This procedure was added to IBM i V5R4M0.

Parameters

Archive_Option

(Input) The type of operation to perform for the DB2 for i Health Center activity detail.

The supported values are:

- 1 = Query only, no archive action is taken
- 2 = Archive only
- 3 = Create archive and archive
- 4 = Query the archive

Note: Option 1 produces a new result set. Options 2 and 3 simply use the results from the last Query option. Option 3 fails if the archive exists.

Refresh_Current_Values

(Input) This option directs how the archive operation is done. This option is only valid with archive options 2 and 3.

The supported values are:

- 0 = No. Indicates that we capture the activity on the entire set of specified schemas and objects.
- 1 = Yes. Indicates that we only refresh the activity of the objects previously captured (based on the short names).
- 2 = None. Use the results from the prior call. A call must have been performed in this job before using this option

Object_Schema

(Input) The target schema or schemas for this operation. A single schema name can be entered. The ‘%' character can be used to direct the procedure to process all schemas with names that start with the same characters which appear before the '%'. When this parameter contains only the '%' character, the procedure processes all schemas within the database.

This name also affects the items refreshed if Refresh_Current_Values = 1.

Object_Name
(Input) The target object name for this operation. Only the '%' character is treated as a wildcard since an underscore is a valid character in a name. The name must be delimited, if necessary, and case sensitive.

This name also affects the items refreshed if Refresh_Current_Values = 1.

Number_Objects_Limit_to_Archive
(Input) The number of objects to save for each design limit.

Number_Of_Limits_Archive
(Input) The number of rows to save per object design limit.

The archive can be used to recognize trends over time. To have meaningful historical comparisons, choose the row count size carefully. This argument is ignored if the Archive_Option is 1 or 4.

Limit_Schema
(Input) The schema that contains the database limit archive.

This argument is ignored if the Archive_Option is 1.

Limit_Table
The table that contains the database limit archive.

This argument is ignored if the Archive_Option is 1.

Authorities

To query an existing archive, *USE object authority is required for the Limit_Schema and Limit_Table. To create an archive, *CHANGE object authority is required for the Limit_Schema. To add to an archive, *CHANGE object authority is required for the Limit_Table.

When Archive_Option is 1 or 3, *USE object authority is required for the Object_Schema and for any objects which are indicated by Object_Name. When an object is encountered and the caller does not have *USE object authority, an SQL0462 warning is placed in the job log. The object is skipped and not included in the procedure result set.

Result Set

When Archive_Option is 1 or 4, a single result set is returned.

The format of the result is as follows. All these items were added for IBM i V5R4M0.

QSYS2.Health_Design_Limits() result set format:

```
"TIMESTAMP" TIMESTAMP NOT NULL,
LIMIT VARCHAR(2000) ALLOCATE(20) DEFAULT NULL,
CURRENT_VALUE FOR COLUMN "VALUE" BIGINT DEFAULT NULL,
PERCENT DECIMAL(5, 2) DEFAULT NULL,
OBJECT_SCHEMA FOR COLUMN BSCHEMA VARCHAR(128) ALLOCATE(10) DEFAULT NULL,
OBJECT_NAME FOR COLUMN BNAME VARCHAR(128) ALLOCATE(20) DEFAULT NULL,
OBJECT_TYPE FOR COLUMN BTYPE VARCHAR(24) ALLOCATE(10) DEFAULT NULL,
SYSTEM_OBJECT_SCHEMA FOR COLUMN SYS_DNAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
SYSTEM_OBJECT_NAME FOR COLUMN SYS_ONAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
PARTITION_NAME FOR COLUMN MBRNAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL
MAXIMUM_VALUE FOR COLUMN "MAXVALUE" BIGINT DEFAULT NULL
LIMIT_ID INTEGER DEFAULT NULL
```

```
LABEL ON COLUMN <result set>
( "TIMESTAMP" IS 'Timestamp',
    LIMIT IS 'Limit',
    CURRENT_VALUE IS 'Current             Value',
    PERCENT IS 'Percent',
    OBJECT_SCHEMA IS 'Object              Schema',
    OBJECT_NAME IS 'Object              Name',
```

```
    OBJECT_TYPE IS 'Object              Type',
    SYSTEM_OBJECT_SCHEMA IS 'System              Object              Schema',
    SYSTEM_OBJECT_NAME IS 'System              Object              Name',
    PARTITION_NAME IS 'Partition           Name',
    MAXIMUM_VALUE IS 'Maximum             Value',
    LIMIT_ID IS 'Limit               ID');
```

Limit Detail

The supported Database Health Center Design limits can be seen on any machine by executing this query:

```
SELECT * FROM QSYS2.SQL_SIZING WHERE SIZING_ID BETWEEN 16000 AND 16999;
```

Table 33. Design limits over objects within a schema.

SIZING_ID	SIZING_NAME	SUPPORTED_VALUE
16100	MAXIMUM NUMBER OF MEMBERS	327670
16101	MAXIMUM NUMBER OF RECORD FORMATS	32
16800	MAXIMUM JOURNAL RECEIVER SIZE	1.09951E+12 (~1 TB)
16801	TOTAL SQL STATEMENTS	0
16802	TOTAL ACTIVE SQL STATEMENTS	0
16803	MAXIMUM SQL PACKAGE SIZE	520093696 (~500 MB)
16804	MAXIMUM LARGE SQL PACKAGE SIZE	1056964608 (~1 GB)
16805	MAXIMUM SQL PROGRAM ASSOCIATED SPACE SIZE	16777216

Error Messages

Table 34. Error messages

Message ID	Error Message Text
SQL0462 W	This warning appears in the job log if the procedure encounters objects for which the user does not have *USE object authority. The warning is provided as an indication that the procedure was unable to process all available objects.

Usage Notes

None

Related Information

None

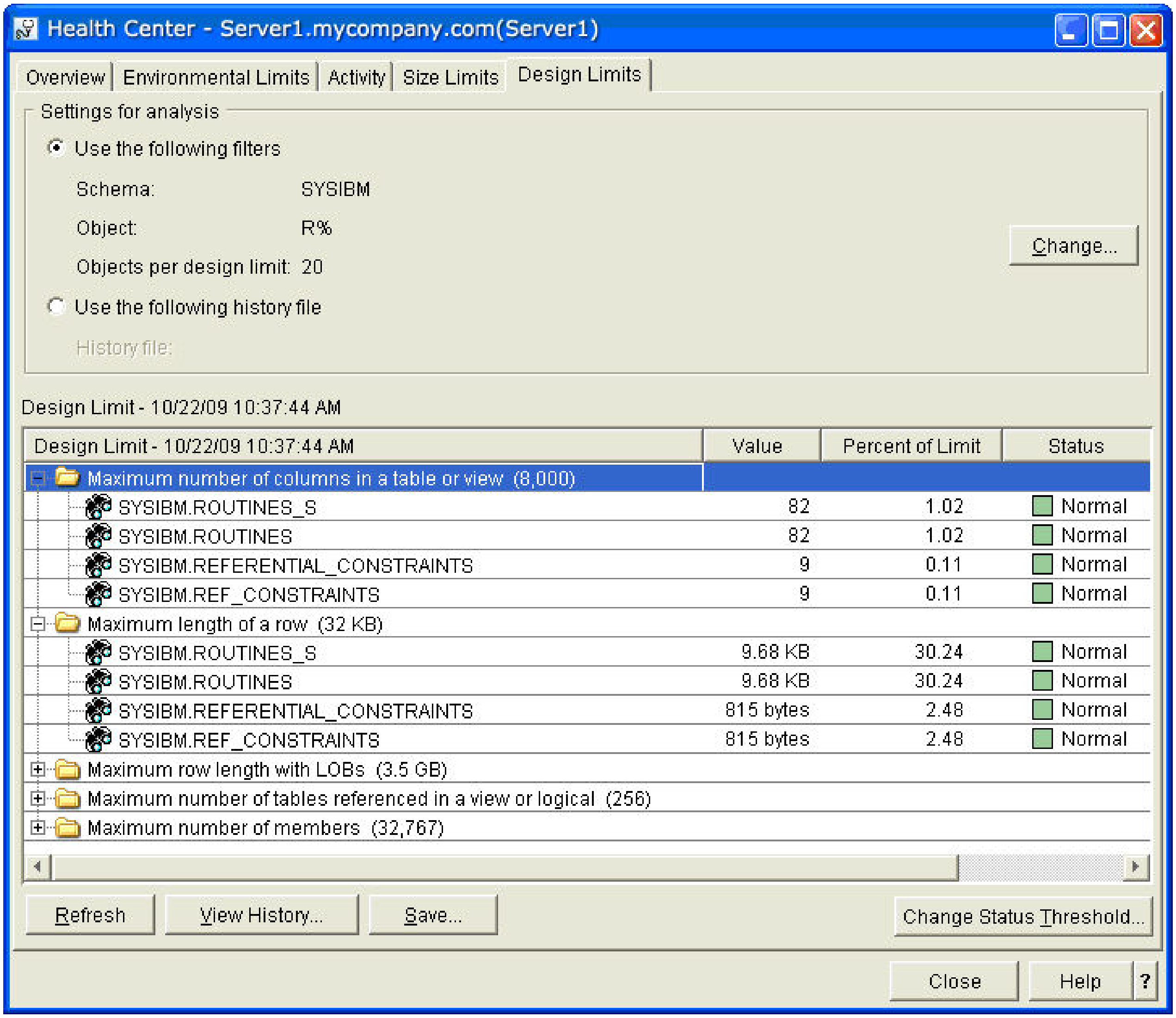
Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 645.

Retrieve the design limit information for all object names which start with the letter R, within the SYSIBM schema, using a maximum of 20 objects per each design limit.

```
CALL QSYS2.Health_Design_Limits(1, 0, 'SYSIBM', 'R%', 20, NULL, NULL, NULL);
```

Example results in System i Navigator:



QSYS2.Health_Size_Limits ()

The QSYS2.Health_Size_Limits () procedure returns detailed size information for database objects within one or more schemas. Size limits help you understand trends towards reaching a database limit such as 'Maximum size of the data in a table partition'.

Procedure definition:

```
CREATE PROCEDURE QSYS2.HEALTH_SIZE_LIMITS(
       IN ARCHIVE_OPTION INTEGER,
       IN REFRESH_CURRENT_VALUES INTEGER,
       IN OBJECT_SCHEMA VARCHAR(258),
       IN OBJECT_NAME VARCHAR(258),
       IN NUMBER_OBJECTS_LIMIT_TO_ARCHIVE INTEGER,
       IN NUMBER_OF_LIMITS_ARCHIVE INTEGER,
       IN LIMIT_SCHEMA VARCHAR(258),
       IN LIMIT_TABLE VARCHAR(258))
       DYNAMIC RESULT SETS 1
       LANGUAGE C
       SPECIFIC QSYS2.HEALTH_SIZE_LIMITS
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'QSYS/QSQHEALTH(SIZE)'
       PARAMETER STYLE SQL;
```

Service Program Name: QSYS/QSQHEALTH

Default Public Authority: *USE

Threadsafe: Yes

IBM i release

This procedure was added to IBM i V5R4M0.

Parameters

Archive_Option
(Input) The type of operation to perform for the DB2 for i Health Center activity detail.

The supported values are:

- 1 = Query only, no archive action is taken
- 2 = Archive only
- 3 = Create archive and archive
- 4 = Query the archive

Note: Option 1 produces a new result set. Options 2 and 3 simply use the results from the last Query option. Option 3 fails if the archive exists.

Refresh_Current_Values
(Input) This option directs how the archive operation is done. This option is only valid with archive options 2 and 3.

The supported values are:

- 0 = No. Indicates that we capture the activity on the entire set of specified schemas and objects.
- 1 = Yes. Indicates that we only refresh the activity of the objects previously captured (based on the short names).
- 2 = None. Use the results from the prior call. A call must have been performed in this job before using this option

Object_Schema
(Input) The target schema or schemas for this operation. A single schema name can be entered. The ‘%' character can be used to direct the procedure to process all schemas with names that start with the same characters which appear before the ‘%'. When this parameter contains only the ‘%' character, the procedure processes all schemas within the database.

This name also affects the items refreshed if Refresh_Current_Values = 1.

Object_Name
(Input) The target object name for this operation. Only the ‘%' character is treated as a wildcard since an underscore is a valid character in a name. The name must be delimited, if necessary, and case sensitive.

This name also affects the items refreshed if Refresh_Current_Values = 1.

Number_Objects_Limit_to_Archive
(Input) The number of objects to save for each size limit.

Number_Of_Limits_Archive
(Input) The number of rows to save per object size limit.

The archive can be used to recognize trends over time. To have meaningful historical comparisons, choose the row count size carefully. This argument is ignored if the Archive_Option is 1 or 4.

Limit_Schema
(Input) The schema that contains the database activity archive.

This argument is ignored if the Archive_Option is 1.

Limit_Table
The table that contains the database activity archive.

This argument is ignored if the Archive_Option is 1.

Authorities

To query an existing archive, *USE object authority is required for the Limit_Schema and Limit_Table. To create an archive, *CHANGE object authority is required for the Limit_Schema. To add to an archive, *CHANGE object authority is required for the Limit_Table.

When Archive_Option is 1 or 3, *USE object authority is required for the Object_Schema and for any objects which are indicated by Object_Name. When an object is encountered and the caller does not have *USE object authority, an SQL0462 warning is placed in the job log. The object is skipped and not included in the procedure result set.

Result Set

When Archive_Option is 1 or 4, a single result set is returned.

The format of the result is as follows. All these items were added for IBM i V5R4M0.

QSYS2.Health_Size_Limits() result set format:

```
"TIMESTAMP" TIMESTAMP NOT NULL,
LIMIT VARCHAR(2000) ALLOCATE(20) DEFAULT NULL,
CURRENT_VALUE FOR COLUMN "VALUE" BIGINT DEFAULT NULL,
PERCENT DECIMAL(5, 2) DEFAULT NULL, OBJECT_SCHEMA FOR COLUMN BSCHEMA VARCHAR(128) ALLOCATE(10) DEFAULT
NULL,
OBJECT_NAME FOR COLUMN BNAME VARCHAR(128) ALLOCATE(20) DEFAULT NULL,
OBJECT_TYPE FOR COLUMN BTYPE VARCHAR(24) ALLOCATE(10) DEFAULT NULL,
SYSTEM_OBJECT_SCHEMA FOR COLUMN SYS_DNAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
SYSTEM_OBJECT_NAME FOR COLUMN SYS_ONAME VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
MAXIMUM_VALUE FOR COLUMN "MAXVALUE" BIGINT DEFAULT NULL,
LIMIT_ID INTEGER DEFAULT NULL,
PARTITION_NAME FOR COLUMN MBRNAME    VARCHAR(10) ALLOCATE(10) DEFAULT NULL,
"SCHEMA" VARCHAR(258) ALLOCATE(10) DEFAULT NULL,
OBJECT VARCHAR(258) ALLOCATE(10) DEFAULT NULL,
"REFRESH" INTEGER DEFAULT NULL
```

```
LABEL ON COLUMN <result set>
( "TIMESTAMP" IS 'Timestamp',
    LIMIT IS 'Limit',
    CURRENT_VALUE IS 'Current             Value',
    PERCENT IS 'Percent',
    OBJECT_SCHEMA IS 'Object              Schema',
    OBJECT_NAME IS 'Object              Name',
    OBJECT_TYPE IS 'Object              Type',
    SYSTEM_OBJECT_SCHEMA IS 'System              Object              Schema',
    SYSTEM_OBJECT_NAME IS 'System              Object              Name',
    MAXIMUM_VALUE IS 'Maximum             Value',
    LIMIT_ID IS 'Limit               ID',
    PARTITION_NAME IS 'Partition           Name',
    "SCHEMA" IS 'Schema              Mask',
    OBJECT IS 'Object              Mask',
    "REFRESH" IS 'Refresh');
```

Limit Detail

The supported Database Health Center Size limits can be seen on any machine by executing this query:

```
SELECT * FROM QSYS2.SQL_SIZING WHERE SIZING_ID BETWEEN 15000 AND 15999;
```

Note: MAXIMUM NUMBER OF OVERFLOW ROWS was added in IBM i 7.1.

Table 35. Size limit information for database objects within a schema.

SIZING_ID	SIZING_NAME	SUPPORTED_VALUE
15000	MAXIMUM NUMBER OF ALL ROWS	4.29E+09
15001	MAXIMUM NUMBER OF VALID ROWS	4.29E+09

Table 35. Size limit information for database objects within a schema. (continued)		
SIZING_ID	**SIZING_NAME**	**SUPPORTED_VALUE**
15002	MAXIMUM NUMBER OF DELETED ROWS	4.29E+09
15003	MAXIMUM TABLE PARTITION SIZE	1.7E+12
15004	MAXIMUM NUMBER OF OVERFLOW ROWS	4.29E+09
15101	MAXIMUM ROW LENGTH	32766
15102	MAXIMUM ROW LENGTH WITH LOBS	3.76E+09
15103	MAXIMUM NUMBER OF PARTITIONS	256
15150	MAXIMUM NUMBER OF REFERENCED TABLES	256
15300	MAXIMUM NUMBER OF TRIGGERS	300
15301	MAXIMUM NUMBER OF CONSTRAINTS	300
15302	MAXIMUM LENGTH OF CHECK CONSTRAINT	2097151
15400	MAXIMUM *MAX4GB INDEX SIZE	4.29E+09
15401	MAXIMUM *MAX1TB INDEX SIZE	1.7E+12
15402	MAXIMUM NUMBER OF INDEX ENTRIES	0
15500	MAXIMUM KEY COLUMNS	120
15501	MAXIMUM KEY LENGTH	32767
15502	MAXIMUM NUMBER OF PARTITIONING KEYS	120
15700	MAXIMUM NUMBER OF FUNCTION PARAMETERS	255
15701	MAXIMUM NUMBER OF PROCEDURE PARAMETERS	1024

Error Messages

Table 36. Error messages	
Message ID	**Error Message Text**
SQL0462 W	This warning appears in the job log if the procedure encounters objects for which the user does not have *USE object authority. The warning is provided as an indication that the procedure was unable to process all available objects.

Usage Notes

None

Related Information

None

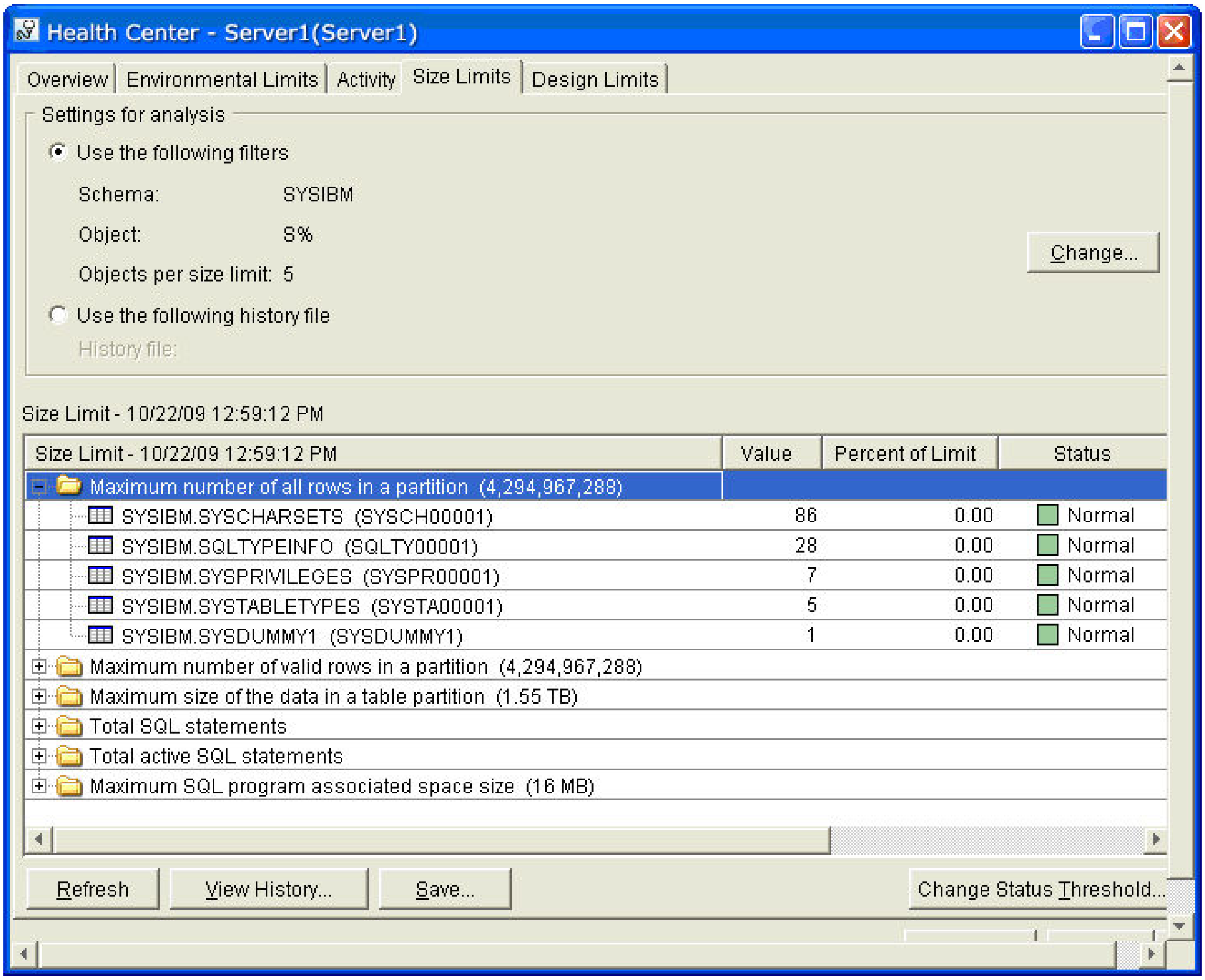
Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 645.

Retrieve the size limit information for all object names which start with the letter S, within the SYSIBM schema, using a maximum of five objects per each design limit.

```
CALL QSYS2.Health_Size_Limits(1, 0, 'SYSIBM', 'S%', 5, NULL, NULL, NULL);
```

Example results in System i Navigator:



QSYS2.Health_Environmental_Limits ()

The QSYS2.Health_Environmental_Limits() procedure returns detail on the top 10 jobs on the system, for different SQL or application limits. The jobs do not have to be in existence. The top 10 information is maintained within DB2 for i and gets reset when the machine is IPLed, the IASP is varied ON, or when the QSYS2.Reset_Environmental_Limits() procedure is called.

Procedure definition:

```
CREATE PROCEDURE QSYS2.HEALTH_ENVIRONMENTAL_LIMITS(
       IN ARCHIVE_OPTION INTEGER,
       IN NUMBER_OF_LIMITS_ARCHIVE INTEGER,
       IN LIMIT_SCHEMA VARCHAR(258),
       IN LIMIT_TABLE VARCHAR(258))
       DYNAMIC RESULT SETS 1
       LANGUAGE C
       SPECIFIC QSYS2.HEALTH_ENVIRONMENTAL_LIMITS
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
       EXTERNAL NAME 'QSYS/QSQHEALTH(ENVIRON)'
       PARAMETER STYLE SQL;
```

Service Program Name: QSYS/QSQHEALTH

Default Public Authority: *USE

Threadsafe: Yes

IBM i release

This procedure was added to IBM i 6.1.

Parameters

Archive_Option
(Input) The type of operation to perform for the DB2 for i Health Center activity detail.

The supported values are:

- 1 = Query only, no archive action is taken
- 2 = Archive only
- 3 = Create archive and archive
- 4 = Query the archive

Note: Option 1 produces a new result set. Options 2 and 3 simply use the results from the last Query option. Option 3 fails if the archive exists.

Number_Of_Limits_Archive
(Input) The number of rows to save per object health limit.

The archive can be used to recognize trends over time. To have meaningful historical comparisons, choose the row count size carefully. This argument is ignored if the Archive_Option is 1 or 4.

Limit_Schema
(Input) The schema that contains the database activity archive.

This argument is ignored if the Archive_Option is 1.

Limit_Table
The table that contains the database activity archive.

This argument is ignored if the Archive_Option is 1.

Authorities

To query an existing archive, *USE object authority is required for the Limit_Schema and Limit_Table. To create an archive, *CHANGE object authority is required for the Limit_Schema. To add to an archive, *CHANGE object authority is required for the Limit_Table.

When Archive_Option is 1 or 3, *USE object authority is required for the Object_Schema and for any objects which are indicated by Object_Name. When an object is encountered and the caller does not have *USE object authority, an SQL0462 warning is placed in the job log. The object is skipped and not included in the procedure result set.

Result Set

When Archive_Option is 1 or 4, a single result set is returned.

The format of the result is as follows. All these items were added for IBM i 6.1.

QSYS2.Health_Environmental_Limits() result set format:

```
"TIMESTAMP" TIMESTAMP NOT NULL,
LIMIT VARCHAR(2000) ALLOCATE(20) DEFAULT NULL,
HIGHWATER_MARK_VALUE FOR COLUMN HIMARK BIGINT DEFAULT NULL,
WHEN_VALUE_WAS_RECORDED FOR COLUMN TIMEHIT TIMESTAMP NOT NULL,
PERCENT DECIMAL(5, 2) DEFAULT NULL,
JOB_NAME VARCHAR(28) ALLOCATE(20) DEFAULT NULL,
"CURRENT_USER" FOR COLUMN CUSER VARCHAR(128) ALLOCATE(10) DEFAULT NULL,
```

```
JOB_TYPE VARCHAR(26) ALLOCATE(20) DEFAULT NULL,
MAXIMUM_VALUE FOR COLUMN MAXVAL BIGINT DEFAULT NULL,
JOB_STATUS VARCHAR(13) DEFAULT NULL,
CLIENT_WRKSTNNAME FOR COLUMN "WRKSTNNAME" VARCHAR(255) DEFAULT NULL,
CLIENT_APPLNAME FOR COLUMN "APPLNAME" VARCHAR(255) DEFAULT NULL,
CLIENT_ACCTNG FOR COLUMN "ACCTNG" VARCHAR(255) DEFAULT NULL,
CLIENTROGRAMID FOR COLUMN "PROGRAMID" VARCHAR(255) DEFAULT NULL,
CLIENT_USERID FOR COLUMN "USERID" VARCHAR(255) DEFAULT NULL,
WHEN_LIMITS_ESTABLISHED FOR COLUMN TIMESET TIMESTAMP NOT NULL,
INTERFACE_NAME FOR COLUMN INTNAME VARCHAR(127) ALLOCATE(10) DEFAULT NULL,
INTERFACE_TYPE FOR COLUMN INTTYPE VARCHAR(63) ALLOCATE(10) DEFAULT NULL,
INTERFACE_LEVEL FOR COLUMN INTLEVEL VARCHAR(63) ALLOCATE(10) DEFAULT NULL,
LIMIT_ID INTEGER DEFAULT NULL
```

```
LABEL ON COLUMN <result set>
( "TIMESTAMP" IS 'Timestamp',
    LIMIT IS 'Limit',
    HIGHWATER_MARK_VALUE IS 'Largest             Value',
    WHEN_VALUE_WAS_RECORDED IS 'Timestamp           When                Recorded',
    PERCENT IS 'Percent',
    JOB_NAME IS 'Job                 Name',
    "CURRENT_USER" IS 'Current             User',
    JOB_TYPE IS 'Job                 Type',
    MAXIMUM_VALUE IS 'Maximum             Value',
    JOB_STATUS IS 'Job                 Status',
    CLIENT_WRKSTNNAME IS 'Client              Workstation         Name',
    CLIENT_APPLNAME IS 'Client              Application         Name',
    CLIENT_ACCTNG IS 'Client              Accounting          Code',
    CLIENTROGRAMID IS 'Client              Program             Identifier',
    CLIENT_USERID IS 'Client              User                Identifier',
    WHEN_LIMITS_ESTABLISHED IS 'Timestamp           Limits              Established',
    INTERFACE_NAME IS 'Interface           Name' ,
    INTERFACE_TYPE IS 'Interface           Type',
    INTERFACE_LEVEL IS 'Interface           Level',
    LIMIT_ID IS 'Limit               ID' );
```

Limit Detail

The supported Database Health Center Environmental limits can be seen on any machine by executing this query:

```
SELECT * FROM QSYS2.SQL_SIZING WHERE SIZING_ID BETWEEN 18200 AND 18299;
```

Note: The **bold** row was added in IBM i 7.1.

Table 37. SQL environmental limits.

SIZING_ID	SIZING_NAME	SUPPORTED_VALUE
18200	MAXIMUM NUMBER OF LOB or XML LOCATORS PER JOB	16000000
18201	MAXIMUM NUMBER OF LOB or XML LOCATORS PER SERVER JOB	209000
18202	MAXIMUM NUMBER OF ACTIVATION GROUPS	0
18203	MAXIMUM NUMBER OF DESCRIPTORS	0
18204	MAXIMUM NUMBER OF CLI HANDLES	160000
18205	MAXIMUM NUMBER OF SQL OPEN CURSORS	21754
18206	MAXIMUM NUMBER OF SQL PSEUDO OPEN CURSORS	0
18207	MAXIMUM LENGTH OF SQL STATEMENT2097152	2097152

Error Messages

None

Usage Notes

None

Related Information

None

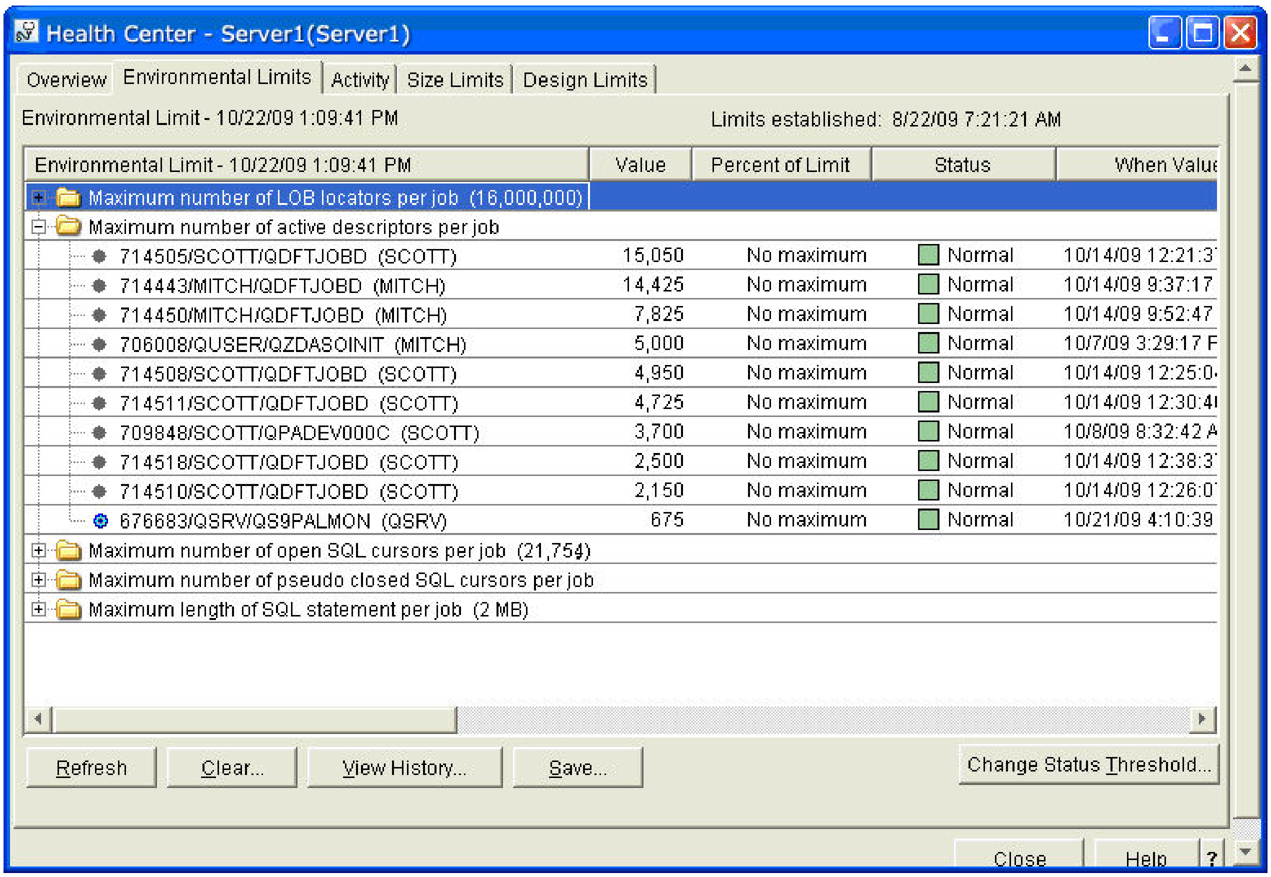
Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 645.

Retrieve the SQL environmental limits for the current database.

```
CALL QSYS2.Health_Environmental_Limits(1, 0, NULL, NULL);
```

Example results in System i Navigator:



QSYS2.Reset_Environmental_Limits ()

The QSYS2.Reset_Environmental_Limits () procedure clears out the environment limit cache for the database. If IASPs are being used, this procedure clears the environment limit cache for the IASP within which it is called.

Procedure definition:

```
CREATE PROCEDURE QSYS2.RESET_ENVIRONMENTAL_LIMITS(
       LANGUAGE C
       SPECIFIC QSYS2.RESET_ENVIRONMENTAL_LIMITS
       NOT DETERMINISTIC
       MODIFIES SQL DATA
       CALLED ON NULL INPUT
```

```
EXTERNAL NAME 'QSYS/QSQSSUDF(RESETENV)'
PARAMETER STYLE SQL;
```

Service Program Name: QSYS/QSQHEALTH

Default Public Authority: *USE

Threadsafe: Yes

IBM i release

This procedure was added to IBM i 6.1.

Parameters

None.

Authorities

This procedure requires the user to have *JOBCTL user special authority or be authorized to the QIBM_DB_SQLADM Function through Application Administration in System i Navigator. The Change Function Usage (CHGFCNUSG) command can also be used to allow or deny use of the function.

For example:

```
CHGFCNUSG FCNID(QIBM_DB_SQLADM) USER(xxxxx) USAGE(*ALLOWED)
```

Result Set

None.

Error Messages

Table 38. Error messages

Message ID	Error Message Text
SQL0552	Not authorized to PROCEDURE.

Usage Notes

None

Related Information

None

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 645.

Reset the SQL environmental limits for the current database.

```
CALL QSYS2.RESET_ENVIRONMENTAL_LIMITS;
```

Monitoring your queries using the Database Monitor

`Start Database Monitor (STRDBMON)` command gathers information about a query in real time and stores this information in an output table. This information can help you determine whether your system and your queries are performing well, or whether they need fine-tuning. Database monitors can generate significant CPU and disk storage overhead when in use.

You can gather performance information for a specific query, for every query on the system, or for a group of queries on the system. When a job is monitored by multiple monitors, each monitor is logging rows to a different output table. You can identify rows in the output database table by its unique identification number.

When you start a monitor using the **`Start Database Monitor (STRDBMON)`** command, the monitor is automatically registered with System i Navigator and appears in the System i Navigator monitor list.

Note: Database monitors also contain the SQL statement text and variable values. If the variable values or SQL statements contain sensitive data you should create database monitors in a library that is not publicly authorized to prevent exposure to the sensitive data.

What kinds of statistics you can gather

The database monitor provides the same information that is provided with the query optimizer debug messages (**`Start Debug (STRDBG)`**) and the **`Print SQL information (PRTSQLINF)`** command. The following is a sampling of the additional information that is gathered by the database monitors:

- System and job name
- SQL statement and subselect number
- Start and end timestamp
- Estimated processing time
- Total rows in table queried
- Number of rows selected
- Estimated number of rows selected
- Estimated number of joined rows
- Key columns for advised index
- Total optimization time
- Join type and method
- ODP implementation

How you can use performance statistics

You can use these performance statistics to generate various reports. For instance, you can include reports that show queries that:

- Use an abundance of the system resources.
- Take a long time to execute.
- Did not run because of the query governor time limit.
- Create a temporary index during execution
- Use the query sort during execution
- Might perform faster with the creation of a keyed logical file containing keys suggested by the query optimizer.

Note: A query that is canceled by an end request generally does not generate a full set of performance statistics. However, it does contain all the information about how a query was optimized, except for runtime or multi-step query information.

Related information

Start Debug (STRDBG) command
Print SQL Information (PRTSQLINF) command
Start Database Monitor (STRDBMON) command

Start Database Monitor (STRDBMON) command

The **Start Database Monitor (STRDBMON)** command starts the collection of database performance statistics for a specified job, for all jobs on the system or for a selected set of jobs. The statistics are placed in a user-specified database table and member. If the table or member do not exist, one is created based on the QAQQDBMN table in library QSYS. If the table and member do exist, the record format of the specified table is verified to insure it is the same.

For each monitor started using the STRDBMON command, the system generates a monitor ID that can be used to uniquely identify each individual monitor. The monitor ID can be used on the ENDDBMON command to uniquely identify which monitor is to be ended. The monitor ID is returned in the informational message CPI436A which is generated for each occurrence of the STRDBMON command. The monitor ID can also be found in column QQC101 of the QQQ3018 database monitor record.

Informally there are two types of monitors. A private monitor is a monitor over one, specific job (or the current job). Only one (1) monitor can be started on a specific job at a time. For example, STRDBMON JOB(*) followed by another STRDBMON JOB(*) within the same job is not allowed. A public monitor is a monitor which collects data across multiple jobs. There can be a maximum of 10 public monitors active at any one time. For example, STRDBMON JOB(*ALL) followed by another STRDBMON JOB(*ALL) is allowed providing the maximum number of public monitors does not exceed 10. You could have 10 public monitors and 1 private monitor active at the same time for any specific job.

If multiple monitors specify the same output file, only one copy of the database statistic records is written to the file for each job. For example, STRDBMON OUTFILE(LIB/TABLE1) JOB(*) and STRDBMON OUTFILE(LIB/TABLE1) JOB(*ALL) target the same output file. For the current job, there are not two copies of the database statistic records–one copy for the private monitor and one copy for the public monitor. There is only one copy of the database statistic records.

If the monitor is started on all jobs (a public monitor), any jobs waiting on queues or started during the monitoring period are included in the monitor data. If the monitor is started on a specific job (a private monitor) that job must be active in the system when the command is issued. Each job in the system can be monitored concurrently by one private monitor and a maximum of 10 public monitors.

The STRDBMON command allows you to collect statistic records for a specific set or subset of the queries running on any job. This filtering can be performed over the job name, user profile, query table names, query estimated run time, TCP/IP address, or any combination of these filters. Specifying a STRDBMON filter helps minimize the number of statistic records captured for any monitor.

Example 1: Starting Public Monitoring

```
STRDBMON   OUTFILE(QGPL/FILE1)  OUTMBR(MEMBER1 *ADD)
 JOB(*ALL)  FRCRCD(10))
```

This command starts database monitoring for all jobs on the system. The performance statistics are added to the member named MEMBER1 in the file named FILE1 in the QGPL library. 10 records are held before being written to the file.

Example 2: Starting Private Monitoring

```
STRDBMON   OUTFILE(*LIBL/FILE3)  OUTMBR(MEMBER2)
 JOB(134543/QPGMR/DSP01)  FRCRCD(20)
```

This command starts database monitoring for job number 134543. The job name is DSP01 and was started by the user named QPGMR. The performance statistics are added to the member named MEMBER2 in the file named FILE3. 20 records are held before being written to the file.

Example 3: Starting Private Monitoring to a File in a Library in an Independent ASP

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(134543/QPGMR/DSP01)
```

This command starts database monitoring for job number 134543. The job name is DSP01 and was started by the user named QPGMR. The performance statistics are added to the member name

DBMONFILE (since OUTMBR was not specified) in the file named DBMONFILE in the library named LIB41. This library could exist in more than one independent auxiliary storage pool (ASP); the library in the name space of the originator's job is always used.

Example 4: Starting Public Monitoring For All Jobs That Begin With 'QZDA

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*ALL/*ALL/QZDA*)
```

This command starts database monitoring for all jobs that whose job name begins with 'QZDA'. The performance statistics (monitor records) are added to member DBMONFILE (since OUTMBR was not specified) in file DBMONFILE in library LIB41. This library could exist in more than one independent auxiliary storage pool (ASP); the library in the name space of the originator's job is always used.

Example 5: Starting Public Monitoring and Filtering SQL Statements That Run Over 10 Seconds

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*ALL)  RUNTHLD(10)
```

This command starts database monitoring for all jobs. Monitor records are created only for those SQL statements whose estimated run time meets or exceeds 10 seconds.

Example 6: Starting Public Monitoring and Filtering SQL Statements That Have an Estimated Temporary Storage Over 200 MB

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*ALL)  STGTHLD(200)
```

This command starts database monitoring for all jobs. Monitor records are created only for those SQL statements whose estimated temporary storage meets or exceeds 200 MB.

Example 7: Starting Private Monitoring and Filtering Over a Specific File

```
STRDBMON   OUTFILE(LIB41/DBMONFILE) JOB(*)
 FTRFILE(LIB41/TABLE1)
```

This command starts database monitoring for the current job. Monitor records are created only for those SQL statements that use file LIB41/TABLE1.

Example 8: Starting Private Monitoring for the Current User

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*)  FTRUSER(*CURRENT)
```

This command starts database monitoring for the current job. Monitor records are created only for those SQL statements that are executed by the current user.

Example 9: Starting Public Monitoring For Jobs Beginning With 'QZDA' and Filtering Over Run Time and File

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*ALL/*ALL/QZDA*)
 RUNTHLD(10)  FTRUSER(DEVLPR1)  FTRFILE(LIB41/TTT*)
```

This command starts database monitoring for all jobs whose job name begins with 'QZDA'. Monitor records are created only for those SQL statements that meet all the following conditions:

- The estimated run time, as calculated by the query optimizer, meets, or exceeds 10 seconds
- Was executed by user 'DEVLPR1'.
- Use any file whose name begins with 'TTT' and resides in library LIB41.

Example 10: Starting Public Monitoring and Filtering SQL Statements That Have Internet Address '9.10.111.77'.

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*ALL)
 FTRINTNETA('9.10.111.77')
```

This command starts database monitoring for all jobs. Monitor records are created only for TCP/IP database server jobs that are using the client IP version 4 address of '9.10.111.77'.

Example 11: Starting Public Monitoring and Filtering SQL Statements That Have a Port Number of 8471

```
STRDBMON   OUTFILE(LIB41/DBMONFILE)  JOB(*ALL)  FTRLCLPORT(8471)
```

This command starts database monitoring for all jobs. Monitor records are created only for TCP/IP database server jobs that are using the local port number 8471.

Example 12: Starting Public Monitoring Based on Feedback from the Query Governor

```
CHGSYSVAL QQRYTIMLMT(200)
 STRDBMON   OUTFILE(LIB41/DBMONFILE) JOB(*ALL) FTRQRYGOVR(*COND)
```

This commands starts database monitoring for all jobs whose estimated run time is expected to exceed 200 seconds, based on the response to the query governor. In this example, data is collected only if the query is canceled or a return code of 2 is returned by a query governor exit program. The query can be canceled by a user response to the inquiry message CPA4259, issued because the query exceeded the query governor limits. It can also be canceled by the program logic inside the registered query governor exit program.

Example 13: Collecting database monitor for Interactive SQL use

```
STRDBMON OUTFILE(QGPL/STRSQLMON1) OUTMBR(*FIRST *REPLACE)
   JOB(*ALL/*ALL/*ALL) TYPE(*DETAIL)
            FTRCLTPGM(STRSQL)
```

This command uses the database monitor pre-filter by Client Special Register Program ID to collect monitor records for all the SQL statements executed by Interactive SQL (STRSQL command) usage.

Related information

Start Database Monitor (STRDBMON) command

End Database Monitor (ENDDBMON) command

The **End Database Monitor (ENDDBMON)** command ends the collection of database performance statistics for a specified job, all jobs on the system, or a selected set of jobs (for example, a generic job name).

To end a monitor, you can specify the job or the monitor ID or both. If only the JOB parameter is specified, the monitor that was started using the same exact JOB parameter is ended - if there is only one monitor which matches the specified JOB. If more than one monitor is active which matches the specified JOB, then the user uniquely identifies which monitor is to be ended by use of the MONID parameter.

When only the MONID parameter is specified, the specified MONID is compared to the monitor ID of the monitor for the current job and to the monitor ID of all active public monitors (monitors that are open across multiple jobs). The monitor matching the specified MONID is ended.

The monitor ID is returned in the informational message CPI436A. This message is generated for each occurrence of the STRDBMON command. Look in the job log for message CPI436A to find the system generated monitor ID, if needed. The monitor ID can also be found in column QQC101 of the QQQ3018 database monitor record.

Restrictions

- If a specific job name and number or JOB(*) was specified on the **Start Database Monitor (STRDBMON)** command, the monitor can only be ended by specifying the same job name and number or JOB(*) on the ENDDBMON command.
- If JOB(*ALL) was specified on the **Start Database Monitor (STRDBMON)** command, the monitor can only be ended by specifying ENDDBMON JOB(*ALL). The monitor cannot be ended by specifying ENDDBMON JOB(*).

When monitoring is ended for all jobs, all the jobs on the system are triggered to close the database monitor output table. However, the ENDDBMON command can complete before all the monitored jobs have written their final statistic records to the log. Use the **Work with Object Locks (WRKOBJLCK)** command to determine that all the monitored jobs no longer hold locks on the database monitor output table before assuming that the monitoring is complete.

Example 1: End Monitoring for a Specific Job

```
ENDDBMON   JOB(*)
```

This command ends database monitoring for the current job.

Example 2: End Monitoring for All Jobs

```
ENDDBMON   JOB(*ALL)
```

This command ends the monitor open across all jobs on the system. If more than one monitor with JOB(*ALL) is active, then the MONID parameter must also be specified to uniquely identify which specific public monitor to end.

Example 3: End Monitoring for an Individual Public Monitor with MONID Parameter

```
ENDDBMON   JOB(*ALL) MONID(061601001)
```

This command ends the monitor that was started with JOB(*ALL) and that has a monitor ID of 061601001. Because there were multiple monitors started with JOB(*ALL), the monitor ID must be specified to uniquely identify which monitor that was started with JOB(*ALL) is to be ended.

Example 4: End Monitoring for an Individual Public Monitor with MONID Parameter

```
ENDDBMON   MONID(061601001)
```

This command performs the same function as the previous example. It ends the monitor that was started with JOB(*ALL) or JOB(*) and that has a monitor ID of 061601001.

Example 5: End Monitoring for All JOB(*ALL) Monitors

```
ENDDBMON   JOB(*ALL/*ALL/*ALL) MONID(*ALL)
```

This command ends all monitors that are active across multiple jobs. It does not end any monitors open for a specific job or the current job.

Example 6: End Monitoring for a Generic Job

```
ENDDBMON   JOB(QZDA*)
```

This command ends the monitor that was started with JOB(QZDA*). If more than one monitor with JOB(QZDA*) is active, then the MONID parameter must also be specified to uniquely identify which individual monitor to end.

Example 7: End Monitoring for an Individual Monitor with a Generic Job

```
ENDDBMON   JOB(QZDA*) MONID(061601001)
```

This command ends the monitor that was started with JOB(QZDA*) and has a monitor ID of 061601001. Because there were multiple monitors started with JOB(QZDA*), the monitor ID must be specified to uniquely identify which JOB(QZDA*) monitor is to be ended.

Example 8: End Monitoring for a Group of Generic Jobs

```
ENDDBMON   JOB(QZDA*) MONID(*ALL)
```

This command ends all monitors that were started with JOB(QZDA*).

Related information

End Database Monitor (ENDDBMON) command

Database monitor performance rows

The rows in the database table are uniquely identified by their row identification number. The information within the file-based monitor (**Start Database Monitor (STRDBMON)**) is written out based upon a set of logical formats which are defined in the database monitor formats. These views correlate closely to the debug messages and the **Print SQL Information (PRSQLINF)** messages.

The database monitor formats section also identifies which physical columns are used for each view and what information it contains. You can use the views to identify the information that can be extracted from the monitor. These rows are defined in several different views which are not shipped with the system and must be created by the user, if wanted. The views can be created with the SQL DDL. The column descriptions are explained in the tables following each figure.

Related concepts

Database monitor formats
This section contains the formats used to create the database monitor SQL tables and views.

Database monitor examples

The System i Navigator interface provides a powerful tool for gathering and analyzing performance monitor data using database monitor. However, you might want to do your own analysis of the database monitor files.

Suppose you have an application program with SQL statements and you want to analyze and performance tune these queries. The first step in analyzing the performance is collection of data. The following examples show how you might collect and analyze data using **Start Database Monitor (STRDBMON)** and **End Database Monitor (ENDDBMON)** commands. Performance data is collected in LIB/PERFDATA for an application running in your current job. The following sequence collects performance data and prepares to analyze it.

1. **STRDBMON** FILE(LIB/PERFDATA) TYPE(*DETAIL). If this table does not exist, the command creates one from the skeleton table in QSYS/QAQQDBMN.
2. Run your application
3. **ENDDBMON**
4. Create views over LIB/PERFDATA using the SQL DDL. Creating the views is not mandatory. All the information resides in the base table that was specified on the **STRDBMON** command. The views simply provide an easier way to view the data.

You are now ready to analyze the data. The following examples give you a few ideas on how to use this data. You must closely study the physical and logical view formats to understand all the data being collected. Then you can create queries that give the best information for your applications.

Related information

Start Database Monitor (STRDBMON) command
End Database Monitor (ENDDBMON) command

Application with table scans example

Determine which queries in your SQL application are implemented with table scans. The complete information can be obtained by joining two views: QQQ1000, which contains information about the SQL statements, and QQQ3000, which contains data about queries performing table scans.

The following SQL query can be used:

```
SELECT (B.End_Timestamp - B.Start_Timestamp) AS TOT_TIME, A.System_Table_Schema,
A.System_Table_Name,
   A.Index_Advised, A.Table_Total_Rows, C.Number_Rows_Returned, A.Estimated_Rows_Selected,
   B.Statement_Text_Long
   FROM LIB.QQQ3000 A, LIB.QQQ1000 B, LIB.QQQ3019 C
   WHERE A.Join_Column = B.Join_Column
   AND A.Join_Column = C.Join_Column
```

Sample output of this query is shown in the following table. Key to this example are the join criteria:

```
 WHERE A.Join_Column = B.Join_Column
   AND A.Join_Column = C.Join_Column
```

Much data about many queries is contained in multiple rows in table LIB/PERFDATA. It is not uncommon for data about a single query to be contained in 10 or more rows within the table. The combination of defining the logical views and then joining the views together allows you to piece together all the data for a query or set of queries. Column QQJFLD uniquely identifies all queries within a job; column QQUCNT is unique at the query level. The combination of the two, when referenced in the context of the logical views, connects the query implementation to the query statement information.

Table 39. Output for SQL Queries that Performed Table Scans

Lib Name	Table Name	Total Rows	Index Advised	Rows Returned	TOT_ TIME	Statement Text
LIB1	**TBL1**	**20000**	**Y**	**10**	**6.2**	`SELECT * FROM LIB1/TBL1 WHERE FLD1 = 'A'`
LIB1	**TBL2**	**100**	**N**	**100**	**0.9**	`SELECT * FROM LIB1/TBL2`
LIB1	**TBL1**	**20000**	**Y**	**32**	**7.1**	`SELECT * FROM LIB1/TBL1 WHERE FLD1 = 'B' AND FLD2 > 9000`

If the query does not use SQL, the SQL information row (QQQ1000) is not created. Without the SQL information row, it is more difficult to determine which rows in LIB/PERFDATA pertain to which query. When using SQL, row QQQ1000 contains the actual SQL statement text that matches the monitor rows to the corresponding query. Only through SQL is the statement text captured. For queries executed using the OPNQRYF command, the OPNID parameter is captured and can be used to tie the rows to the query. The OPNID is contained in column Open_Id of row QQQ3014.

Queries with table scans example

Like the preceding example that showed which SQL applications were implemented with table scans, the following example shows all queries that are implemented with table scans.

```
SELECT (D.End_Timestamp - D.Start_Timestamp) AS TOT_TIME, A.System_Table_Schema,
A.System_Table_Name,
     A.Table_Total_Rows, A.Index_Advised,
     B.Open_Id, B.Open_Time,
     C.Clock_Time_to_Return_All_Rows, C.Number_Rows_Returned,
     D.Result_Rows, D.Statement_Text_Long
   FROM LIB.QQQ3000 A INNER JOIN LIB.QQQ3014 B
     ON (A.Join_Column = B.Join_Column
     LEFT OUTER JOIN LIB.QQQ3019 C
     ON (A.Join_Column = C.Join_Column)
```

```
     LEFT OUTER JOIN LIB.QQQ1000 D
     ON (A.Join_Column = D.Join_Column)
```

In this example, the output for all queries that performed table scans are shown in the following table.

Note: The columns selected from table QQQ1000 do return NULL default values if the query was not executed using SQL. For this example assume that the default value for character data is blanks and the default value for numeric data is an asterisk (*).

Table 40. Output for All Queries that Performed Table Scans

Lib Name	Table Name	Total Rows	Index Advised	Query OPNID	ODP Open Time	Clock Time	Recs Rtned	Rows Rtned	TOT_ TIME	Statement Text
LIB1	**TBL1**	**20000**	**Y**		**1.1**	**4.7**	**10**	**10**	**6.2**	`SELECT * FROM LIB1/TBL1 WHERE FLD1 = 'A'`
LIB1	**TBL2**	**100**	**N**		**0.1**	**0.7**	**100**	**100**	**0.9**	`SELECT * FROM LIB1/TBL2`
LIB1	**TBL1**	**20000**	**Y**		**2.6**	**4.4**	**32**	**32**	**7.1**	`SELECT * FROM LIB1/TBL1 WHERE FLD1 = 'A' AND FLD2 > 9000`
LIB1	**TBL4**	**4000**	**N**	**QRY04**	**1.2**	**4.2**	**724**	*****	*****	*****

If the SQL statement text is not needed, joining to table QQQ1000 is not necessary. You can determine the total time and rows selected from data in the QQQ3014 and QQQ3019 rows.

Table scan detail example

Your next step could include further analysis of the table scan data. The previous examples contained a column titled Index Advised. A 'Y' (yes) in this column is a hint from the query optimizer that the query could perform better with an index to access the data. For the queries where an index is advised, the rows selected by the query are low in comparison to the total number of table rows. This selectivity is another indication that a table scan might not be optimal. Finally, a long execution time might highlight queries that could be improved by performance tuning.

The next logical step is to look into the index advised optimizer hint. The following query can be used:

```
SELECT A.System_Table_Schema, A.System_Table_Name,
     A.Index_Advised, A.Index_Advised_Columns,
     A.Index_Advised_Columns_Count, B.Open_Id,
     C.Statement_Text_Long
   FROM LIB.QQQ3000 A INNER JOIN LIB.QQQ3014 B
     ON (A.Join_Column = B.Join_Column)
     LEFT OUTER JOIN LIB.QQQ1000 C
     ON (A.Join_Column = C.Join_Column)
   WHERE A.Index_Advised = 'Y'
```

There are two slight modifications from the first example. First, the selected columns have been changed. Most important is the selection of Index_Advised_Columns containing a list of possible key columns to use when creating the suggested index. Second, the query selection limits the output to those table scan queries where the optimizer advises that an index is created (A.Index_Advised = 'Y'). The following table shows what the results might look like.

Table 41. Output with Recommended Key Columns						
Lib Name	**Table Name**	**Index Advised**	**Advised Key columns**	**Advised Primary Key**	**Query OPNID**	**Statement Text**
LIB1	**TBL1**	**Y**	**FLD1**	**1**		`SELECT * FROM LIB1/TBL1 WHERE FLD1 = 'A'`
LIB1	**TBL1**	**Y**	**FLD1, FLD2**	**1**		`SELECT * FROM LIB1/TBL1 WHERE FLD1 = 'B' AND FLD2 > 9000`
LIB1	**TBL4**	**Y**	**FLD1, FLD4**	**1**	**QRY04**	

Determine whether it makes sense to create a permanent index as advised by the optimizer. In this example, creating one index over LIB1/TBL1 satisfies all three queries since each use a primary or left-most key column of FLD1. By creating one index over LIB1/TBL1 with key columns FLD1, FLD2, there is potential to improve the performance of the second query even more. Consider how often these queries are run and the overhead of maintaining an additional index over the table when deciding whether to create the suggested index.

If you create a permanent index over FLD1, FLD2 the next sequence of steps is as follows:

1. Start the performance monitor again
2. Rerun the application
3. End the performance monitor
4. Re-evaluate the data.

It is likely that the three index-advised queries are no longer performing table scans.

Additional database monitor examples

The following are additional ideas or examples on how to extract information from the performance monitor statistics. All the examples assume that data has been collected in LIB/PERFDATA and the documented views have been created.

1. How many queries are performing dynamic replans?

```
SELECT COUNT(*)
  FROM   LIB.QQQ1000
  WHERE  Dynamic_Replan_Reason_Code <> 'NA'
```

2. What is the statement text and the reason for the dynamic replans?

```
SELECT Dynamic_Replan_Reason_Code, Statement_Text_Long
  FROM   LIB.QQQ1000
  WHERE  Dynamic_Replan_Reason_Code <> 'NA'
```

   Note: You need to refer to the description of column Dynamic_Replan_Reason_Code for definitions of the dynamic replan reason codes.

3. How many indexes have been created over LIB1/TBL1?

```
SELECT COUNT(*)
  FROM   LIB.QQQ3002
  WHERE  System_Table_Schema = 'LIB1'
    AND  System_Table_Name = 'TBL1'
```

4. What key columns are used for all indexes created over LIB1/TBL1 and what is the associated SQL statement text?

```
SELECT A.System_Table_Schema, A.System_Table_Name,
     A.Index_Advised_Columns, B.Statement_Text_Long
```

```
    FROM LIB.QQQ3002 A, LIB.QQQ1000 B
    WHERE A.Join_Column = B.Join_Column
      AND A.System_Table_Schema = 'LIB1'
      AND A.System_Table_Name = 'TBL1'
```

Note: This query shows key columns only from queries executed using SQL.

5. What key columns are used for all indexes created over LIB1/TBL1 and what was the associated SQL statement text or query open ID?

```
SELECT A.System_Table_Schema, A.System_Table_Name, A.Index_Advised_Columns,
     B.Open_Id, C.Statement_Text_Long
   FROM LIB.QQQ3002 A INNER JOIN LIB.QQQ3014 B
     ON (A.Join_Column = B.Join_Column)
   LEFT OUTER JOIN LIB.QQQ1000 C
     ON (A.Join_Column = C.Join_Column)
   WHERE A.System_Table_Schema LIKE '%'
     AND A.System_Table_Name = '%'
```

Note: This query shows key columns from all queries on the system.

6. What types of SQL statements are being performed? Which are performed most frequently?

```
SELECT CASE Statement_Function
     WHEN 'O' THEN 'Other'
     WHEN 'S' THEN 'Select'
     WHEN 'L' THEN 'DDL'
     WHEN 'I' THEN 'Insert'
     WHEN 'U' THEN 'Update'
   ELSE 'Unknown'
   END, COUNT(*)
   FROM LIB.QQQ1000
   GROUP BY Statement_Function
   ORDER BY 2 DESC
```

7. Which SQL queries are the most time consuming? Which user is running these queries?

```
SELECT (End_Timestamp - Start_Timestamp), Job_User,
     Current_User_Profile, Statement_Text_Long
   FROM LIB.QQQ1000
   ORDER BY 1 DESC
```

8. Which queries are the most time consuming?

```
SELECT (A.Open_Time + B.Clock_Time_to_Return_All_Rows),
     A.Open_Id, C.Statement_Text_Long
   FROM LIB.QQQ3014 A LEFT OUTER JOIN LIB.QQQ3019 B
     ON (A.Join_Column = B.Join_Column)
   LEFT OUTER JOIN LIB.QQQ1000 C
     ON (A.Join_Column = C.Join_Column)
   ORDER BY 1 DESC
```

Note: This example assumes that detail data was collected (STRDBMON TYPE(*DETAIL)).

9. Show the data for all SQL queries with the data for each SQL query logically grouped.

```
SELECT A.*
   FROM LIB.PERFDATA A, LIB.QQQ1000 B
   WHERE A.QQJFLD = B.Join_Column
```

Note: This might be used within a report that will format the interesting data into a more readable format. For example, all reason code columns can be expanded by the report to print the definition of the reason code. Physical column QQRCOD = 'T1' means that a table scan was performed because no indexes exist over the queried table.

10. How many queries are implemented with temporary tables because a key length greater than 2000 bytes, or more than 120 key columns was specified for ordering?

```
SELECT COUNT(*)
   FROM LIB.QQQ3004
   WHERE Reason_Code = 'F6'
```

11. Which SQL queries were implemented with nonreusable ODPs?

```
SELECT B.Statement_Text_Long
   FROM LIB.QQQ3010 A, LIB.QQQ1000 B
   WHERE A.Join_Column = B.Join_Column
     AND A.ODP_Implementation = 'N';
```

12. What is the estimated time for all queries stopped by the query governor?

```
SELECT Estimated_Processing_Time, Open_Id
   FROM LIB.QQQ3014
   WHERE Stopped_By_Query_Governor = 'Y'
```

Note: This example assumes that detail data was collected (STRDBMON TYPE(*DETAIL)).

13. Which queries estimated time exceeds actual time?

```
SELECT A.Estimated_Processing_Time,
     (A.Open_Time + B.Clock_Time_to_Return_All_Rows),
     A.Open_Id, C.Statement_Text_Long
   FROM LIB.QQQ3014 A LEFT OUTER JOIN LIB.QQQ3019 B
     ON (A.Join_Column = B.Join_Column)
   LEFT OUTER JOIN LIB.QQQ1000 C
     ON (A.Join_Column = C.Join_Column)
   WHERE A.Estimated_Processing_Time/1000 >
     (A.Open_Time + B.Clock_Time_to_Return_All_Rows)
```

Note: This example assumes that detail data was collected (STRDBMON TYPE(*DETAIL)).

14. Should you apply a PTF for queries containing UNIONs? Yes, if any queries are performing UNIONs. Do any of the queries perform this function?

```
 SELECT COUNT(*)
   FROM QQQ3014
   WHERE Has_Union = 'Y'
```

Note: If the result is greater than 0, apply the PTF.

15. You are a system administrator and an upgrade to the next release is planned. You want to compare data from the two releases.
    - Collect data from your application on the current release and save this data in LIB/CUR_DATA
    - Move to the next release
    - Collect data from your application on the new release and save this data in a different table: LIB/NEW_DATA
    - Write a program to compare the results. You need to compare the statement text between the rows in the two tables to correlate the data.

Using System i Navigator with detailed monitors

You can work with detailed monitors from the System i Navigator interface. The detailed SQL performance monitor is the System i Navigator version of the STRDBMON database monitor, found on the native interface.

You can start this monitor by right-clicking SQL Performance Monitors under the Database portion of the System i Navigator tree and selecting **New** > **Monitor**. This monitor saves detailed data in real time to a hard disk. It does not need to be paused or ended in order to analyze the results. You can also choose to run a Visual Explain based on the data gathered by the monitor. Since this monitor saves data in real time, it might have a performance impact on your system.

Starting a detailed monitor

You can start a detailed monitor from the System i Navigator interface.

You can start this monitor by right-clicking **SQL Performance Monitors** under the Database portion of the System i Navigator tree and selecting **New** > **SQL Performance Monitor**.

When you create a detailed monitor, you can filter the information that you want to capture.

Initial number of records:
Select to specify the number of records initially allocated for the monitor. The 'Initial number of records' option is used to pre-allocate storage to the database monitor out file. When collecting large amounts of monitor records, this option improves the collection performance by avoiding automatic storage extensions that occur as a file grows in size.

Minimum estimated query runtime:
Select to include queries that exceed a specified amount of time. Select a number and then a unit of time.

Minimum estimated temporary storage:
Select to include queries that exceed a certain amount of temporary storage. Specify a size in MB.

Job name:
Select to filter by a specific job name. Specify a job name in the field. You can specify the entire ID or use a wildcard. For example, 'QZDAS*' finds all jobs where the name starts with 'QZDAS.'

Job user:
Select to filter by a job user. Specify a user ID in the field. You can specify the entire ID or use a wildcard. For example, 'QUSER*' finds all user IDs where the name starts with 'QUSER.'

Current user:
Select to filter by the current user of the job. Specify a user ID in the field. You can specify the entire ID or use a wildcard. For example, 'QSYS*' finds all users where the name starts with 'QSYS.'

Client location:
Select to filter by Internet access. The input needs to be in IPv4 or IPv6 form.

1. IP version 4 address in dotted decimal form. Specify an Internet Protocol version 4 address in the form nnn.nnn.nnn.nnn where each nnn is a number in the range 0 through 255.
2. IP version 6 address in colon hexadecimal form. Specify an internet protocol version 6 address in the form xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx, where each xxxx is a hex number in the range 0 through FFFF. IP version 6 includes the IPv4-mapped IPv6 address form (for example, ::FFFF:1.2.3.4). For IP version 6, the compressed form of the address is allowed.
3. IP host domain name. Specify an internet host domain name of up to 254 characters in length.

Local port:
Select to filter by port number. You can select a port from the list or else enter your own port number.

Ports in the list include:

- 446 - DRDA/DDM
- 447 - DRDA/DDM
- 448 - Secure DRDA/DDM (SSL)
- 4402 - QXDAEDRSQL server
- 8471 - Database server
- 9471 - Secure database server (SSL)

Query Governor limits:
Select to search for queries that have exceeded or are expected to exceed the query governor limits set for the system. Choose from the following options:

- Always collect information when exceeded
- Conditional collection of information when exceeded

Client registers:
Select to filter by the client register information.

Statements that access these objects:
Select to filter by only queries that use certain tables. Click **Browse** to select tables to include. To remove a table from the list, select the table and click **Remove**. A maximum of 10 table names can be specified.

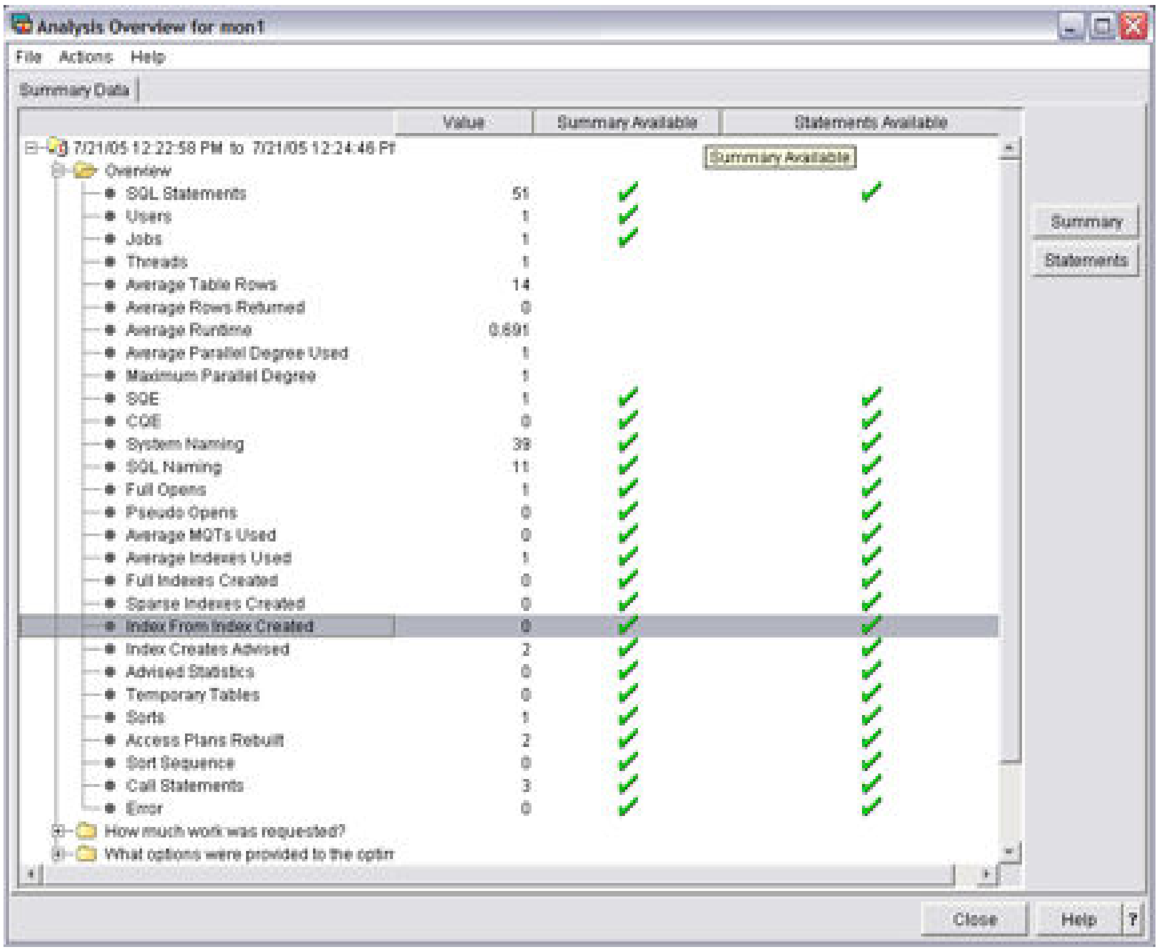
Activity to monitor:
Select to collect monitor output for user-generated queries or for both user-generated and system-generated queries.

You can choose which jobs you want to monitor or choose to monitor all jobs. You can have multiple instances of monitors running on your system at one time. You can create up to 10 detailed monitors to monitor all jobs. When collecting information for all jobs, the monitor will collect on previously started jobs or new jobs that are started after the monitor is created. You can edit this list by selecting and removing jobs from the **Selected jobs** list.

Analyzing detailed monitor data

SQL performance monitors provides several predefined reports that you can use to analyze your monitor data.

To view these reports, right-click a monitor and select **Analyze**. The monitor does not need to be ended in order to view this information.



On the Analysis Overview dialog, you can view overview information or else choose one of the following categories:

- How much work was requested?
- What options were provided to the optimizer?
- What implementations did the optimizer use?
- What types of SQL statements were requested?
- Miscellaneous information
- I/O information

From the Actions menu, you can choose one of the following summary predefined reports:

User summary
Contains a row of summary information for each user. Each row summarizes all SQL activity for that user.

Job summary
Contains a row of information for each job. Each row summarizes all SQL activity for that job. This information can be used to tell which jobs on the system are the heaviest users of SQL. These jobs are perhaps candidates for performance tuning. You could then start a separate detailed performance monitor on an individual job to get more detailed information without having to monitor the entire system.

Operation summary
Contains a row of summary information for each type of SQL operation. Each row summarizes all SQL activity for that type of SQL operation. This information provides the user with a high-level indication of the type of SQL statements used. For example, are the applications mostly read-only, or is there a large amount of update, delete, or insert activity. This information can then be used to try specific performance tuning techniques. For example, if many INSERTs are occurring, you might use an OVRDBF command to increase the blocking factor or the QDBENCWT API.

Program summary
Contains a row of information for each program that performed SQL operations. Each row summarizes all SQL activity for that program. This information can be used to identify which programs use the most or most expensive SQL statements. Those programs are then potential candidates for performance tuning. A program name is only available if the SQL statements are embedded inside a compiled program. SQL statements that are issued through ODBC, JDBC, or OLE DB have a blank program name unless they result from a procedure, function, or trigger.

In addition, when a green check is displayed under Summary column, you can select that row and click **Summary** to view information about that row type. Click **Help** for more information about the summary report. To view information organized by statements, click **Statements**.

Comparing monitor data

You can use System i Navigator to compare data sets in two or more monitors.

System i Navigator provides you with two types of comparison. The first is a simple compare that provides a high-level comparison of the monitors or snapshots. The second is a detailed comparison. The simple compare provides you with enough data about the monitors or snapshots to help you determine if a detailed comparison is helpful.

To launch a simple compare, go to **System i Navigator** > ***system name*** > **SQL performance monitors**. Right-click one or more monitors and select **Compare**.

To launch a detailed comparison, select the Detailed Comparison tab.

On the Detailed Comparison dialog, you can specify information about the data sets that you want to compare.

Name
The name of the monitors that you want to compare.

Schema mask
Select any names that you want the comparison to ignore. For example, consider the following scenario: You have an application running in a test schema and it is optimized. Now you move it to the production schema and you want to compare how it executes there. The statements in the comparison are identical except that the statements in the test schema use "TEST" and the statements in the production schema use "PROD". You can use the schema mask to ignore "TEST" in the first monitor and "PROD" in the second monitor. Then the statements in the two monitors appear identical.

Statements that ran longer than
The minimum runtime for statements to be compared.

Minimum percent difference
The minimum difference in key attributes of the two statements being compared that determines if the statements are considered equal or not. For example, if you select 25% as the minimum percent different, only matching statements whose key attributes differ by 25% or more are returned.

When you click **Compare**, both monitors are scanned for matching statements. Any matches found are displayed side-by-side for comparison of key attributes of each implementation.

On the Comparison output dialog, you view statements that are included in the monitor by clicking **Show Statements**. You can also run Visual Explain by selecting a statement and clicking **Visual Explain**.

Any matches found are displayed side-by-side for comparison of key attributes of each implementation.

Viewing statements in a monitor

You can view SQL statements that are included in a detailed monitor.

Right-click any detailed monitor in the SQL performance monitor window and select **Show statements**.

The filtering options provide a way to focus in on a particular area of interest:

Minimum runtime for the longest execution of the statement:
Select to include statements that exceed a certain amount of time. Select a number and then a unit of time.

Statements that ran on or after this date and time:
Select to include statements run at a specified date and time. Select a date and time.

Statements that reference the following objects:
Select to include statements that use or reference certain objects. Click **Browse** to select objects to include.

Statements that contain the following text:
Select to include only those statements that contain a specific type of SQL statement. For example, specify SELECT if you only want to include statements that are using SELECT. The search is case insensitive for ease of use. For example, the string 'SELECT' finds the same entries as the search string 'select'.

Multiple filter options can be specified. In a multi-filter case, the candidate entries for each filter are computed independently. Only those entries that are present in all the candidate lists are shown. For example, if you specified options **Minimum runtime for the longest execution of the statement** and **Statements that ran on or after this date and time**, you will be shown statements with the minimum runtime that ran on or after the specified date and time.

Related reference

Index advisor
The query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index.

Importing a monitor

You can import monitor data that has been collected using some other interface by using System i Navigator.

Monitors that are created using the **`Start Database Monitor (STRDBMON)`** command are automatically registered with System i Navigator. They are also included in the list of monitors displayed by System i Navigator.

To import monitor data, right-click **SQL Performance monitors** and select **Import**. Once you have imported a monitor, you can analyze the data.

Index advisor

The query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index.

The optimizer is able to perform radix index probe over any combination of the primary key columns, plus one additional secondary key column. Therefore it is important that the first secondary key column is the most selective secondary key column. The optimizer uses radix index scan with any of the remaining secondary key columns. While radix index scan is not as fast as radix index probe, it can still reduce the number of keys selected. It is recommended that secondary key columns that are fairly selective are included.

Determine the true selectivity of any secondary key columns and whether you include those key columns in the index. When building the index, make the primary key columns the left-most key columns, followed by any of the secondary key columns chosen, prioritized by selectivity.

After creating the suggested index and executing the query again, it is possible that the query optimizer will choose not to use the suggested index. It does not include join, ordering, and grouping criteria. The SQE optimizer includes selection, join, ordering, and grouping criteria when suggesting indexes. Local selection advice can now factor in both AND and OR predicates with the qualifications mentioned below.

You can access index advisor information in many different ways. These ways include:

- The index advisor interface in System i Navigator
- SQL performance monitor Show statements
- Visual Explain interface
- Querying the Database monitor view 3020 - Index advised.

Related reference

Overview of information available from Visual Explain
You can use Visual Explain to view many types of information.

Database monitor view 3020 - Index advised (SQE)
Displays the SQL logical view format for database monitor QQQ3020.

Viewing statements in a monitor
You can view SQL statements that are included in a detailed monitor.

Index advice and OR predicates

Index advice generation to handle OR predicates

Index Advisor has been extended to include queries that OR together local selection (WHERE clause) columns over a single table. OR advice requires two or more indexes to be created as a dependent set.

If any of the OR'd indexes are missing, the optimizer won’t be able to cost and choose these dependent indexes for implementation of the OR based query.

This relationship between OR based indexes in the SYSIXADV index advice table is with a new **DEPENDENT_ADVICE_COUNT** column.

Some restrictions with this support:

- OR'd predicate advice appears only if no other advice is generated
- Maximum of 5 predicates OR'd together
- Advised for files with OR'd local selection that get costed in the primary join dial when optimizing a join query

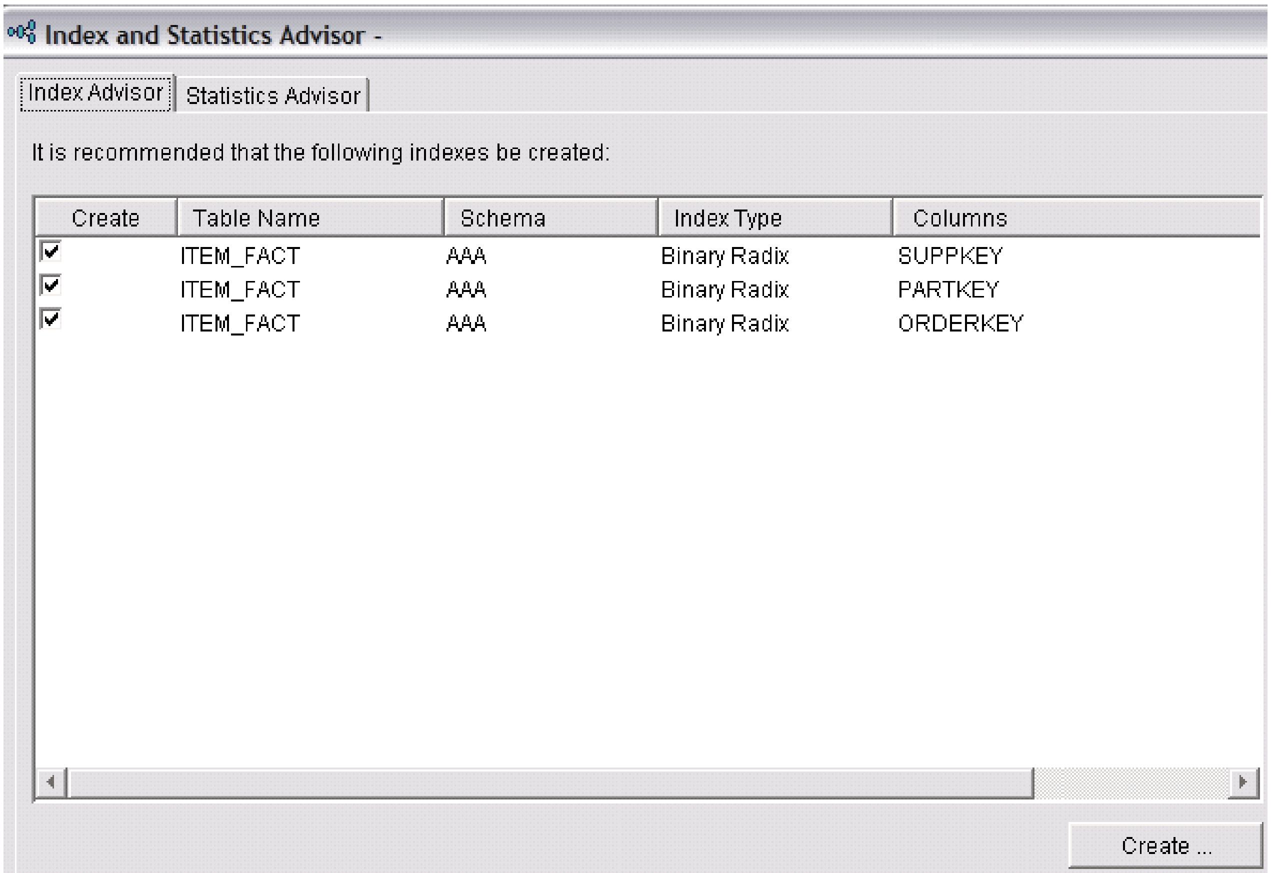
When Index Advisor shows highly dependent advice, use of the exact match capability from Show Statements to find the query in the plan cache is helpful. Once found, use Visual Explain to discover the dependent index advice specific to that query.

Index Or Advice example

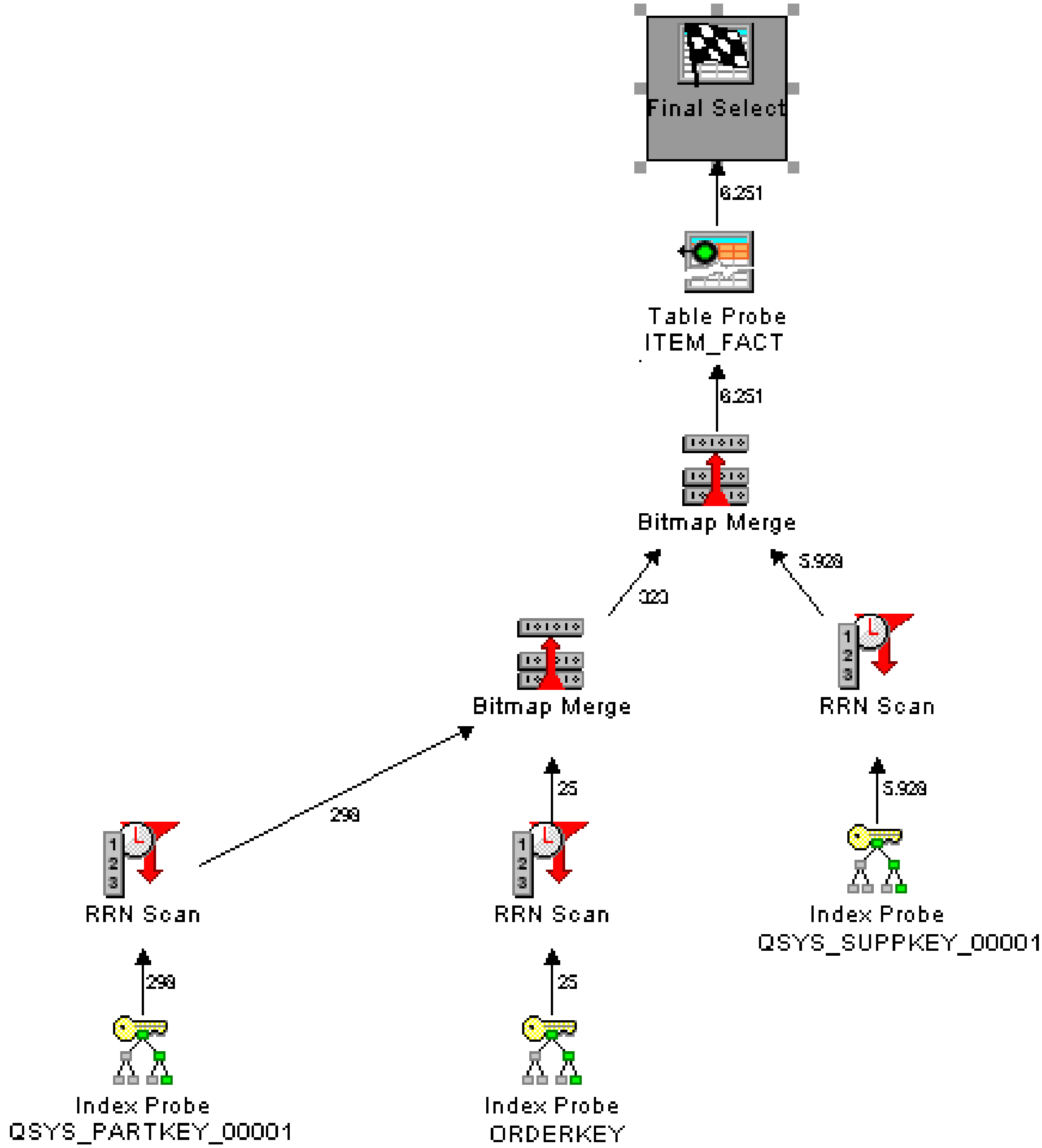
- Should advise indexes over all OR'd predicate columns
- All 3 advised indexes will have **DEPENDENT_ADVICE_COUNT>0**
- Execution with indexes should produce bitmap implementation and register no new advice

```
SELECT orderkey, partkey, suppkey,
linenumber, shipmode orderpriority
FROM item_fact
WHERE OrderKey <= 10 OR
SuppKey <= 10 OR
PartKey <= 10
OPTIMIZE FOR ALL ROWS
```

The graphic below shows the Index advice for the OR'd predicate columns:



The graphic below depicts the Visual Explain showing the implementation of merge bitmap representation using the OR'd advice indexes:



Displaying index advisor information

You can display index advisor information from the optimizer using System i Navigator.

System i Navigator displays information found in the QSYS2/SYSIXADV system table.

To display index advisor information, follow these steps:

1. In the System i Navigator window, expand the system that you want to use.
2. Expand **Databases**.
3. Right-click the database that you want to work with and select **Index Advisor** > **Index Advisor**.

You can also find index advisor information for a specific schema or a specific table by right-clicking on a schema or table object.

Once you have displayed the information, you have several options. You can create an index from the list, remove the index advised from the list, or clear the list entirely. You can also right-click on an index and select **Show SQL**, launching a Run SQL Scripts session with the index creation statement. Finally, you can right-click on an advised index and select Show Statements. With additional information automatically provided in the advised index filter for the Plan Cache search, the resulting SQL statements shown will be a better match to the original queries that generated that specific index advice.

Depending on if you are viewing the index advice at the database level or the schema level your list could be large. Once you have the list displayed, follow these steps to subset your list:

1. Go to the **View** menu option, and select **Customize this view** > **Include ...**.
2. Enter the information you would like to filter the list by.
3. Press the **OK** button to get the refreshed list of index advice.

Database manager indexes advised system table

This topic describes the indexes advised system table.

Table 42. SYSIXADV system table

Column name	System column name	Data type	Description
TABLE_NAME	TBNAME	VARCHAR(258)	Table over which an index is advised
TABLE_SCHEMA	DBNAME	VARCHAR(128)	SQL schema containing the table
SYSTEM_TABLE_NAME	SYS_TNAME	CHAR(10)	System table name on which the index is advised
PARTITION_NAME	TBMEMBER	CHAR(10)	Partition detail for the index
KEY_COLUMNS_ADVISED	KEYSADV	VARCHAR(16000)	Column names for the advised index
LEADING_COLUMN_KEYS	LEADKEYS	VARCHAR(16000)	Leading, Order Independent keys. the keys at the beginning of the KEY_COLUMNS_ADVISED field which could be reordered and still satisfy the index being advised.
INDEX_TYPE	INDEX_TYPE	CHAR(14)	Radix (default) or EVI
LAST_ADVISED	LASTADV	TIMESTAMP	Last time this row was updated
TIMES_ADVISED	TIMESADV	BIGTINT	Number of times this index has been advised
ESTIMATED_CREATION_TIME	ESTTIME	INT	Estimated number of seconds for index creation
REASON_ADVISED	REASON	CHAR(2)	Coded reason why index was advised
LOGICAL_PAGE_SIZE	PAGESIZE	INT	Recommended page size for index
MOST_EXPENSIVE_QUERY	QUERYCOST	INT	Execution time in seconds of the query
AVERAGE_QUERY_ESTIMATE	QUERYEST	INT	Average execution time in seconds of the query
TABLE_SIZE	TABLE_SIZE	BIGINT	Number of rows in table when the index was advised
NLSS_TABLE_NAME	NLSSNAME	CHAR(10)	NLSS table to use for the index
NLSS_TABLE_SCHEMA	NLSSDBNAME	CHAR(10)	Schema name of the NLSS table

Table 42. SYSIXADV system table (continued)			
Column name	**System column name**	**Data type**	**Description**
MTI_USED	MTIUSED	BIGINT	The number of times that this specific Maintained Temporary Index (MTI) has been used by the optimizer. The optimizer stops using a matching MTI once a permanent index is created.
MTI_CREATED	MTICREATED	INTEGER	The number of times that this specific Maintained Temporary Index (MTI) has been created by the optimizer. MTIs do not persist across system IPLs.
LAST_MTI_USED	LASTMTIUSE	TIMESTAMP	The timestamp representing the last time this specific Maintained Temporary Index (MTI) was used by the optimizer to improve the performance of a query. The MTI Last Used field can be blank. The blank field indicates that an MTI which exactly matches this advice has never been used by the queries which generated this index advice.
AVERAGE_QUERY_ESTIMATE_MICRO	QRYMICRO	BIGINT	Average execution time in microseconds of the query which drove the index advice
EVI_DISTINCT_VALUES	EVIVALS	INTEGER	Recommended value to use when creating the advised EVI index. This value is *n* within the WITH *n* DISTINCT VALUES clause on the CREATE INDEX SQL statement.
INCLUDE_COLUMNS	INCLCOL	CLOB(10000)	EVI INCLUDE expressions for index creation.
FIRST_ADVISED	FIRSTADV	TIMESTAMP	When this row was inserted.
SYSTEM_TABLE_SCHEMA	SYS_DNAME	CHAR(10)	System name of the table schema.
MTI_USED_FOR_STATS	MTISTATS	BIGINT	Number of times Maintained Temporary Index was used as a source for optimizer statistics.
LAST_MTI_USED_FOR_STATS	LASTMTISTA	TIMESTAMP	The timestamp representing the last time this specific Maintained Temporary Index was used as a source of statistics by the optimizer to improve the performance of a query.
DEPENDENT_ADVICE_COUNT	DEPCNT	BIGINT	The number of times this index advice was dependent upon other advice.

Index advisor column descriptions

Displays the columns that are used in the Index advisor window.

Table 43. Columns used in Index advisor window

Column name	Description
Table for Which Index was Advised	The optimizer is advising creation of a permanent index over this table. This value is the long name for the table. The advice was generated because the table was queried and no existing permanent index could be used to improve the performance of the query.
Schema	Schema or library containing the table.
System Schema	System name of the schema.
System Name	System table name on which the index is advised
Partition	Partition detail for the index. Possible values: • <blank>, which means For all partitions • For Each Partition • specific name of the partition
Keys Advised	Column names for the advised index. The order of the column names is important. The names are listed in the same order as in the CREATE INDEX SQL statement. An exception is when the leading, order independent key information indicates that the ordering can be changed.
Leading Keys Order Independent	Leading, Order Independent keys. the keys at the beginning of the KEY_COLUMNS_ADVISED field which could be reordered and still satisfy the index being advised.
Index Type Advised	Radix (default) or EVI
Last Advised for Query Use	The timestamp representing the last time this index was advised for a query.
Times Advised for Query Use	The cumulative number of times this index has been advised. This count ceases to increase once a matching permanent index is created. The row of advice remains in this table until the user removes it
Estimated Index Creation Time	Estimated time in seconds to create this index.
Reason advised	Reason why index was advised. Possible values are: Row selection Ordering/Grouping Row selection and Ordering/Grouping
Logical Page Size Advised (KB)	Recommended page size to be used on the PAGESIZE keyword of the CREATE INDEX SQL statement when creating this index.
Most Expensive Query Estimate	Execution time in seconds of the longest running query which generated this index advice.
Average of Query Estimates	Average execution time in seconds of all queries that generated this index advice.

Table 43. Columns used in Index advisor window (continued)	
Column name	**Description**
Rows in Table when Advised	Number of rows in table for the last time this index was advised.
NLSS Table Advised	The sort sequence table in use by the query which generated the index advice. For more detail on sort sequences:
NLSS Schema Advised	The schema of the sort sequence table.
MTI Used	The number of times that this specific Maintained Temporary Index (MTI) has been used by the optimizer.
MTI Created	The number of times that this specific Maintained Temporary Index (MTI) has been created by the optimizer. MTIs do not persist across system IPLs.
MTI Last Used	The timestamp representing the last time this specific Maintained Temporary Index (MTI) was used by the optimizer to improve the performance of a query. The MTI Last Used field can be blank. A blank field indicates that an MTI which exactly matches this advice has never been used by the queries which generated this index advice.
EVI Distinct Values	Recommended value to use when creating the advised EVI index. This value is **n** within the WITH **n** DISTINCT VALUES clause on the CREATE INDEX SQL statement.
First Advised	The date/time when a row is first added to the Index Advisor table for this advice.
MTI Used for Stats	The number of times that this specific Maintained Temporary Index (MTI) has been used by the optimizer.
MTI Last Used for Stats	The timestamp representing the last time this specific Maintained Temporary Index (MTI) was used as a source of statistics by the optimizer to improve the performance of a query. The MTI Last Used field can be blank.
Dependent Advice Count	Dependent implies that this advised index is dependent on the creation of other dependent advised indexes and all of the other dependent indexes must be created in order for a index ORing bitmap implementation can be costed and utilized. • Zero - this advised index stands on its own, no OR selection • Greater than Zero – Compare this column against the TIMES_ADVISED column to understand how often this advised index has both OR and non-OR selection.

Querying database monitor view 3020 - Index advised
The index advisor information can be found in the Database Monitor view 3020 - Index advised (SQE).

The advisor information is stored in columns QQIDXA, QQIDXK, and QQIDXD. When the QQIDXA column contains a value of 'Y' the optimizer is advising you to create an index using the key columns shown in column QQIDXD. The intention of creating this index is to improve the performance of the query.

In the list of key columns contained in column QQIDXD, the optimizer has listed what it considers the suggested primary and secondary key columns. Primary key columns are columns that can significantly reduce the number of keys selected based on the corresponding query selection. Secondary key columns are columns that might or might not significantly reduce the number of keys selected.

Column QQIDXK contains the number of suggested primary key columns that are listed in column QQIDXD. These primary key columns are the left-most suggested key columns. The remaining key columns are considered secondary key columns and are listed in order of expected selectivity based on the query. For example, assuming QQIDXK contains the value of four and QQIDXD specifies seven key columns, then the first four key columns are the primary key columns. The remaining three key columns are the suggested secondary key columns.

Condensing index advice

Many times, the index advisor advises several different indexes for the same table. You can condense these advised indexes into the best matches for your queries.

1. In the **System i Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Right-click the database that you want to work with and select **Index Advisor** > **Condense Advised Indexes**.

Depending on if you are viewing the condensed index advice at the database level or the schema level your list could be large. Once you have the list displayed, follow these steps to subset your list:

1. Go to the **View** menu option, and select **Customize this view** > **Include ...**
2. Enter the information you would like to filter the list by.
3. Select **OK** to get the refreshed list of condensed index advice.

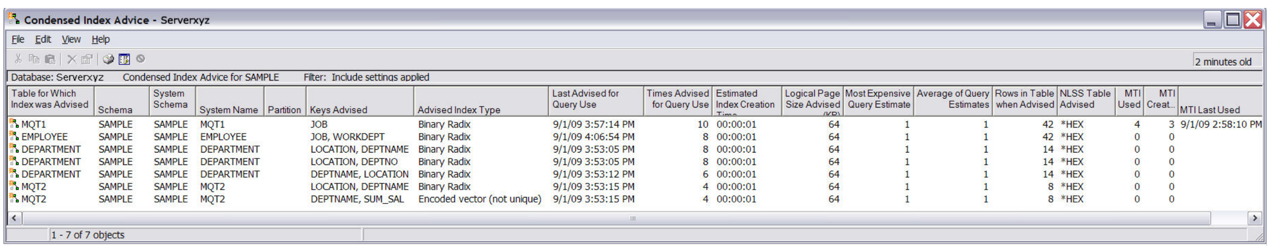


Table for Which Index was Advised	Schema	System Schema	System Name	Partition	Keys Advised	Advised Index Type	Last Advised for Query Use	Times Advised for Query Use	Estimated Index Creation	Logical Page Size Advised	Most Expensive Query Estimate	Average of Query Estimates	Rows in Table when Advised	NLSS Table Advised	MTI Used	MTI Creat...	MTI Last Used
MQT1	SAMPLE	SAMPLE	MQT1		JOB	Binary Radix	9/1/09 3:57:14 PM	10	00:00:01	64	1	1	42	*HEX	4	3	9/1/09 2:58:10 PM
EMPLOYEE	SAMPLE	SAMPLE	EMPLOYEE		JOB, WORKDEPT	Binary Radix	9/1/09 4:06:54 PM	8	00:00:01	64	1	1	42	*HEX	0	0	
DEPARTMENT	SAMPLE	SAMPLE	DEPARTMENT		LOCATION, DEPTNAME	Binary Radix	9/1/09 3:53:05 PM	8	00:00:01	64	1	1	14	*HEX	0	0	
DEPARTMENT	SAMPLE	SAMPLE	DEPARTMENT		LOCATION, DEPTNO	Binary Radix	9/1/09 3:53:05 PM	8	00:00:01	64	1	1	14	*HEX	0	0	
DEPARTMENT	SAMPLE	SAMPLE	DEPARTMENT		DEPTNAME, LOCATION	Binary Radix	9/1/09 3:53:12 PM	6	00:00:01	64	1	1	14	*HEX	0	0	
MQT2	SAMPLE	SAMPLE	MQT2		LOCATION, DEPTNAME	Binary Radix	9/1/09 3:53:15 PM	4	00:00:01	64	1	1	8	*HEX	0	0	
MQT2	SAMPLE	SAMPLE	MQT2		DEPTNAME, SUM_SAL	Encoded vector (not unique)	9/1/09 3:53:15 PM	4	00:00:01	64	1	1	8	*HEX	0	0	

Viewing your queries with Visual Explain

You can use the **Visual Explain** tool with System i Navigator to create a query graph that graphically displays the implementation of an SQL statement. You can use this tool to see information about both static and dynamic SQL statements. Visual Explain supports the following types of SQL statements: SELECT, INSERT, UPDATE, and DELETE.

Queries are displayed using a graph with a series of icons that represent different operations that occur during implementation. This graph is displayed in the main window. In the lower portion of the pane, the SQL statement that the graph is based on is displayed. If Visual Explain is started from Run SQL Scripts, you can view the debug messages issued by the optimizer by clicking the **Optimizer messages** tab. The query attributes are displayed in the right pane.

Visual Explain can be used to graphically display the implementations of queries stored in the detailed SQL performance monitor. However, it does not work with tables resulting from the memory-resident monitor.

Starting Visual Explain

There are two ways to invoke the Visual Explain tool. The first, and most common, is through System i Navigator. The second is through the Visual Explain (QQQVEXPL) API.

You can start Visual Explain from any of the following windows in System i Navigator:

- Enter an SQL statement in the **Run SQL Scripts** window. Select the statement and choose **Explain** or **Run and Explain** from the **Visual Explain** menu.
- Expand the list of available SQL Performance Monitors. Right-click a detailed SQL Performance Monitor and choose the **Show Statements** option. Select filtering information and select the statement in the List of Statements window. Right-click and select **Visual Explain**. You can also start an SQL

Performance Monitor from Run SQL Scripts. Select **Start SQL Performance monitor** from the **Monitor** menu.

- Start the SQL Details for Jobs function by right-clicking **Databases** and select **SQL Details for Jobs**. Click **Apply**. Select a job from the list and right-click and select **Show Details**. When the SQL is displayed in the lower pane, you can start Visual Explain by right-clicking on **Statement** and selecting **Visual Explain**.
- Right-click SQL Plan Cache and select **Show Statements**. Select filtering information and select the statement in the List of Statements window. Right-click and select **Visual Explain**.
- Expand the list of available SQL Plan Cache Snapshots. Right-click a snapshot and select **Show Statements**. Select filtering information and select the statement in the List of Statements window. Right-click and select **Visual Explain**.
- Expand the list of SQL Plan Cache Event Monitors. Right-click an event monitor and select **Show Statements**. Select filtering information and select the statement in the List of Statements window. Right-click and select **Visual Explain**.

You have three options when running Visual Explain from Run SQL Scripts.

Visual Explain only
: This option allows you to explain the query without actually running it. The data displayed represents the estimate of the query optimizer.

  Note: Some queries might receive an error code 93 stating that they are too complex for displaying in Visual Explain. You can circumvent this error by selecting the "Run and Explain" option.

Run and Explain
: If you select Run and Explain, the query is run by the system before the diagram is displayed. This option might take a significant amount of time, but the information displayed is more complete and accurate.

Explain while running
: For long running queries, you can choose to start Visual Explain while the query is running. By refreshing the Visual Explain diagram, you can view the progress of the query.

In addition, a database monitor table that was not created as a result of using System i Navigator can be explained through System i Navigator. First you must import the database monitor table into System i Navigator. To import, right-click the SQL Performance Monitors and choose the **Import** option. Specify a name for the performance monitor (name it is known by within System i Navigator) and the qualified name of the database monitor table. Be sure to select Detailed as the type of monitor. Detailed represents the file-based (STRDBMON) monitor while Summary represents the memory-resident monitor (which is not supported by Visual Explain). Once the monitor has been imported, follow the steps to start Visual Explain from within System i Navigator.

You can save your Visual Explain information as an SQL Performance monitor. This monitor can be useful if you started the query from Run SQL Scripts and want to save the information for later comparison. Select **Save as Performance monitor** from the **File** menu.

Related information

Visual Explain (QQQVEXPL) API

Overview of information available from Visual Explain

You can use Visual Explain to view many types of information.

The information includes:

- Information about each operation (icon) in the query graph
- Highlight expensive icons
- The statistics and index advisor
- The predicate implementation of the query
- Basic and detailed information in the graph

Information about each operation (icon) in the query graph

As stated before, the icons in the graph represent operations that occur during the implementation of the query. The order of operations is shown by the arrows connecting the icons. If parallelism was used to process an operation, the arrows are doubled. Occasionally, the optimizer "shares" hash tables with different operations in a query, causing the lines of the query to cross.

You can view information about an operation by selecting the icon. Information is displayed in the **Attributes** table in the right pane. To view information about the environment, click an icon and then select **Display query environment** from the **Action** menu. Finally, you can view more information about the icon by right-clicking the icon and selecting **Help**.

Highlight expensive icons

You can highlight problem areas (expensive icons) in your query using Visual Explain. Visual Explain offers you two types of expensive icons to highlight: by processing time or number of rows. You can highlight icons by selecting **Highlight expensive icons** from the **View** menu.

The statistics and index advisor

During the query implementation, the optimizer can determine if statistics need to be created or refreshed, or if an index might make the query run faster. You can view these recommendations using the Statistics and Index Advisor from Visual Explain. Start the advisor by selecting **Advisor** from the **Action** menu. Additionally, you can begin collecting statistics or create an index directly from the advisor.

The predicate implementation of the query

Visual explain allows you to view the implementation of query predicates. Predicate implementation is represented by a blue plus sign next to an icon. You can expand this view by right-clicking the icon and selecting **Expand**. or open it into another window. Click an icon to view attributes about the operation. To collapse the view, right-click anywhere in the window and select **Collapse**. This function is only available on V5R3 or later systems.

The optimizer can also use the Look Ahead Predicate Generation to minimize the random the I/O costs of a join. To highlight predicates that used this method, select **Highlight LPG** from the **View** menu.

Basic and full information in the graph

Visual Explain also presents information in two different views: basic and full. The basic view only shows those icons that are necessary to understand the implementation of the SQL statement. It excludes some preliminary, or intermediate operations that are not essential for understanding the main flow of query implementation. The full view might show more icons that further depict the flow of the execution tree. You can change the graph detail by select **Graph Detail** from the **Options** menu and selecting either **Basic** or **Full**. The default view is **Basic**. In order to see all the detail for a **Full** view, change the Graph Detail to **Full**, close out Visual Explain, and run the query again. The setting for Graph Detail persists.

For more information about Visual Explain and the different options that are available, see the Visual Explain online help.

Refresh the Visual Explain diagram

For long running queries, you can refresh the visual explain graph with runtime statistical information before the query is complete. Refresh also updates the appropriate information in the attributes section of the icon shown on the right of the screen. In order to use the **Refresh** option, you need to select **Explain while Running** from the Run SQL Scripts window.

To refresh the diagram, select **Refresh** from the **View** menu. Or click the **Refresh** button in the toolbar.

Related reference

Index advisor

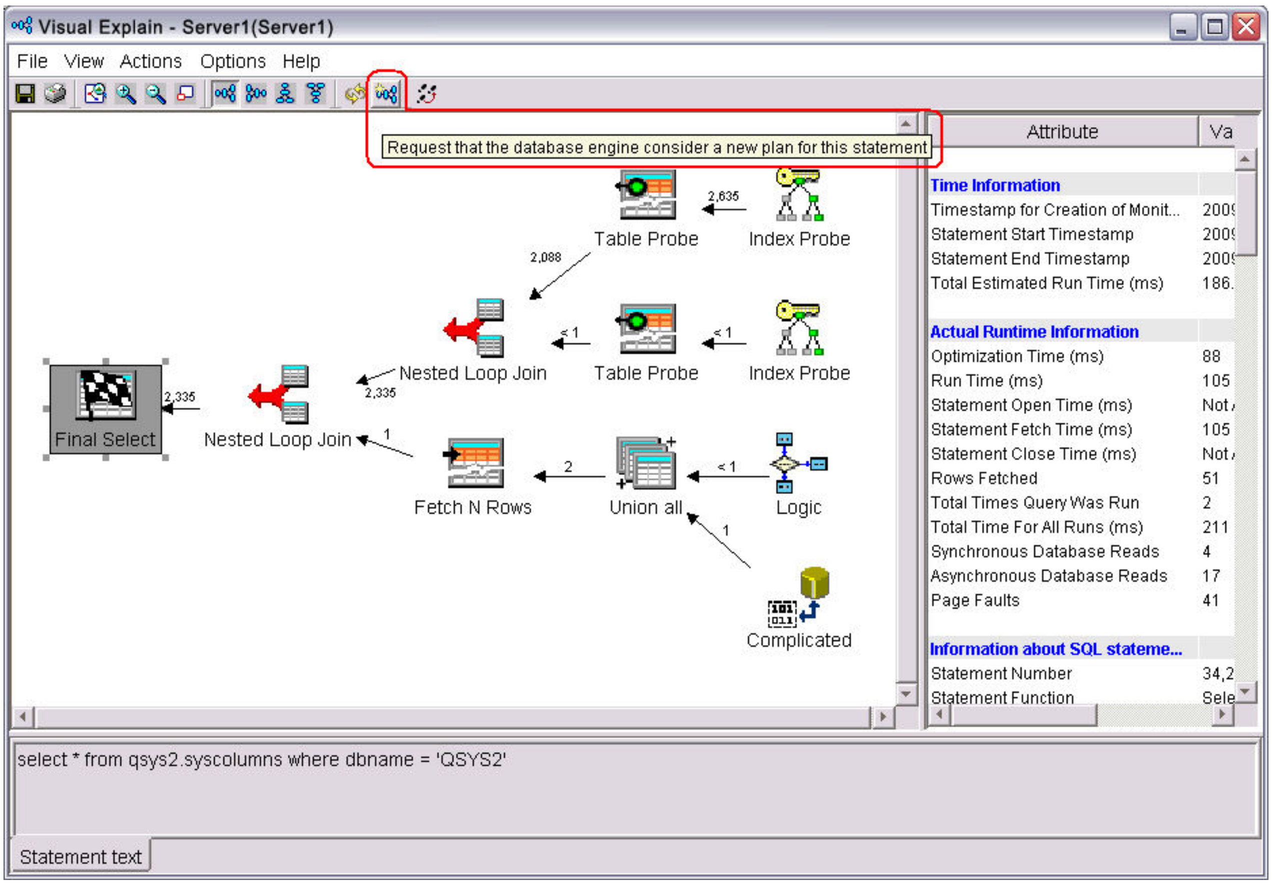
The query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index.

Adaptive Query Processing in Visual Explain
You can use Visual Explain to request a new plan.

There might be times when you are asked to performance tune a query while the query is still running. For instance, a query might be taking a long time to finish. After viewing the plan in Visual Explain, you decide to create the recommended index to improve the speed of the query. So you create the index and then want to signal the database optimizer to consider a new plan based on the new index.

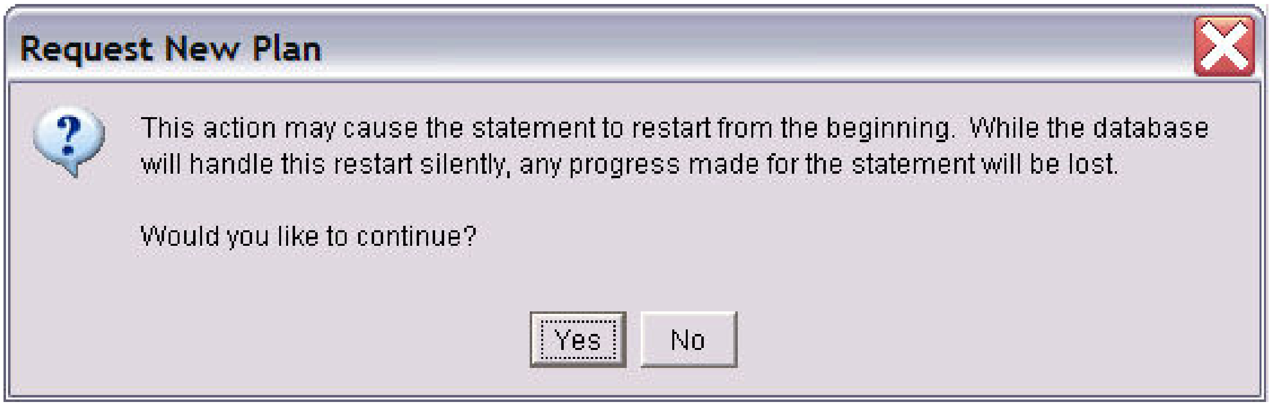
Here are the steps to request the database engine to consider a new plan while running in Visual Explain:

1. Open Run SQL Scripts.
2. Type in a query.
3. Go to the **Visual Explain** menu and select **Explain While Running**.
4. The Visual Explain window is displayed.
5. Next, go to the **Actions** menu and select **Request New Plan**.



A message box appears.

Select **Yes** to restart the query.



The database optimizer considers any changes to the query environment, and determines whether it is appropriate to generate a new plan. It might be possible that the database optimizer decides it is better to continue using the existing plan.

Note: This capability could also be available when selecting Visual Explain of a statement in the SQL Details for a Job window, or the SQL Plan Cache Show Statements window.

Related reference

Adaptive Query Processing
Adaptive Query Processing analyzes actual query run time statistics and uses that information for subsequent optimizations.

Optimizing performance using the Plan Cache

The SQL Plan Cache contains a wealth of information about the SQE queries being run through the database. Its contents are viewable through the System i Navigator GUI interface. Certain portions of the plan cache can also be modified.

In addition, procedures are provided to allow users to programmatically work with the plan cache. These procedures can be invoked using the SQL CALL statement.

The Plan Cache interface provides a window into the database query operations on the system. The interface to the Plan Cache resides under the **System i Navigator** > **system name** > **Database**.

Within the SQL Plan Cache folder are two folders, SQL Plan Cache Snapshots and SQL Plan Cache Event Monitors.

Clicking the SQL Plan Cache Snapshots folder shows a list of any snapshots gathered so far. A snapshot is a database monitor file generated from the plan cache at the time a 'New Snapshot' is requested. It can be treated much the same as the SQL Performance Monitors list. The same analysis capability exists for snapshots as exists for traditional SQL performance monitors.

Clicking the SQL Plan Cache Event Monitors shows a list of any events that have been defined. Plan Cache event monitors, when defined, generate database monitor information from plans as they are being removed from the cache. The list includes currently active events as well as ones that have completed. Like a snapshot, the event monitor is a database monitor file. Consequently, the same analysis capability available to SQL performance monitors and snapshots can be used on the event file.

The plan cache is an actively changing cache. Therefore, it is important to realize that it contains timely information. If information over long periods of time is of interest, an event monitor could be defined to ensure that information is captured on any plans that are removed from the cache over time. Alternatively, you could consider implementing a method of performing periodic snapshots of the plan cache to capture trends and heavy usage periods. See the discussion on IBM supplied, callable SQL procedures later in this section on plan cache.

Note: SQL plan cache snapshots and SQL plan cache event monitors also contain the SQL statement text and variable values. If the variable values or SQL statements contain sensitive data you should create SQL plan cache snapshots and SQL plan cache event monitors in a library that is not publicly authorized to prevent exposure to the sensitive data.

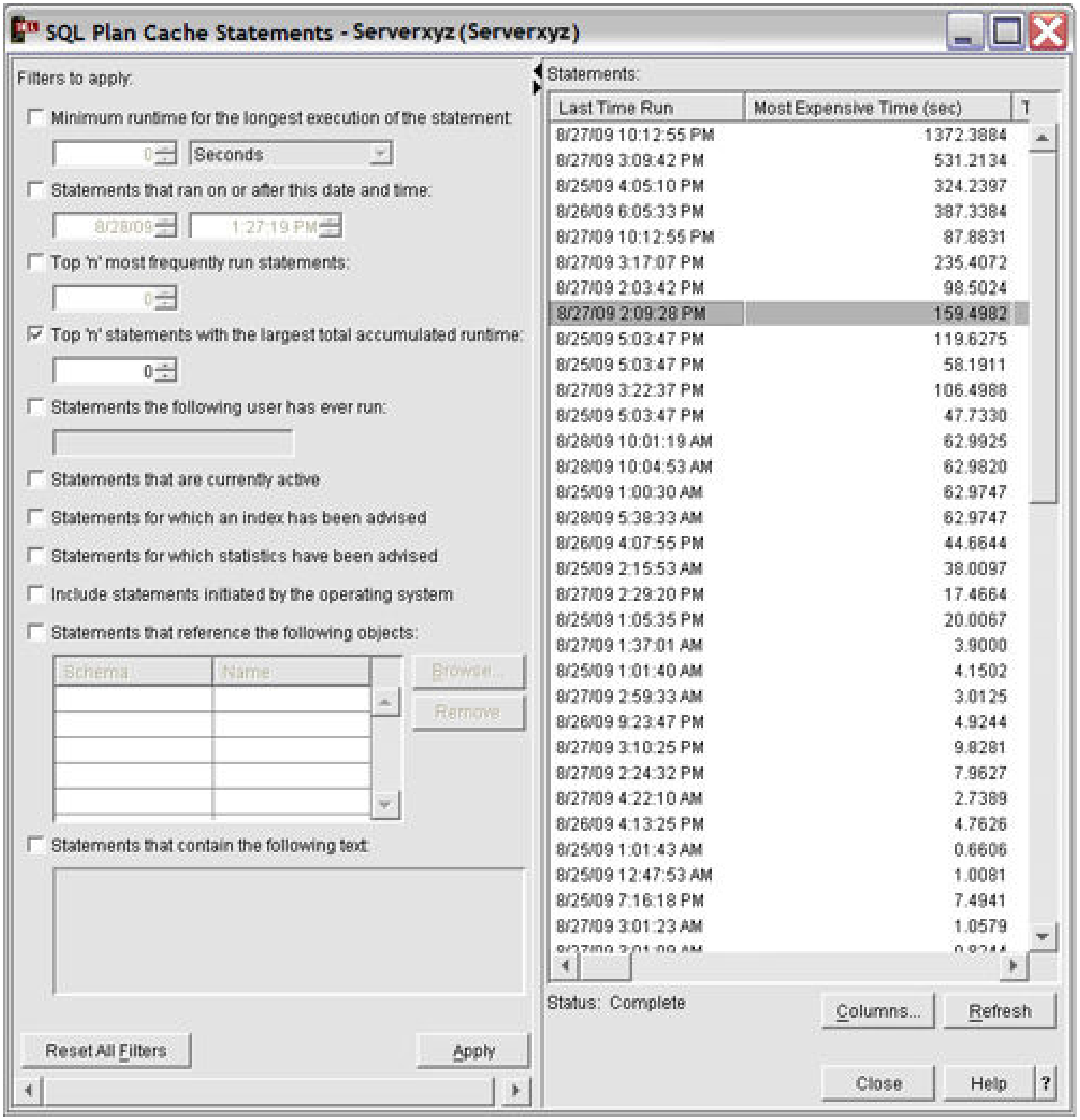
Related concepts

Plan cache

The plan cache is a repository that contains the access plans for queries that were optimized by SQE.

SQL Plan Cache - Show Statements

By right-clicking the SQL Plan Cache icon, a series of options are shown which allow different views of current plan cache of the database. The **SQL Plan Cache** > **Show Statements** option opens a screen with filtering capability. This screen provides a direct view of the current plan cache on the system.



Press the **Apply** or **Refresh** button to display the current Plan Cache statements. The information shown includes the SQL query text, last time the query ran, most expensive single instance run, total processing time consumed, total number of times run, and information about the user and job that first created the plan entry.

The information also includes several per run averages, including average runtime, average result set size and average temporary storage usage. There is an adjusted average processing time which is the average discounting any anomalous runs.

The display also shows how many times, if any, that the database engine resued the results of a prior run, avoiding rerunning the entire statement. There is also a Save Results button (not shown) that allows you to save the statement list, for example, to a .csv file or spreadsheet.

Finally, the numeric identifier and plan score are also displayed. For more detail on the columns displayed, see rzajqcolumnsplancache.dita

Statement Options

By highlighting one or more plans and right clicking, a menu with several possible actions appears.

Visual Explain
Shows a visual depiction of the access plan and provides more detailed performance analysis. Note only one statement can be highlighted when performing this action.

Show Longest Runs
Shows details of up to 10 of the longest running instances of that statement. Within the Longest Runs list, you can right click a statement and select **Visual Explain**, **Work With SQL Statement**, **Work With SQL Statement and Variables**, **Save to New...** snapshot or **Remove**. Snapshots are useful for capturing the information for that specific run in Visual Explain. Removing old or superfluous runs makes room to capture future runs. Only one statement can be highlighted when performing these actions. Any runs removed only affect which runs are shown in the list. The total time, total number of runs, and other information for the statement are still calculated including the runs removed from the list.

Show Active Jobs
Displays a list of jobs on the system that are currently using that statement or statements.

Show User History
Shows a list of all user IDs that have run that statement along with the last time they ran it.

Work with SQL Statement
Displays a scripting window containing the SQL statement. The scripting window is useful for working with and tuning the statement directly, or for just viewing the statement in its own window. Only one statement can be highlighted when performing this action.

Work with SQL Statements and Variables
Displays a scripting window containing the SQL Statement and any parameter markers entered with their specific values for that run of the SQL statement.

Save to New...
Allows you to create a snapshot of the selected statements.

Plan
Right-click to show options for modifying the plan:

Change Plan Score allows you to set the score to a specific value. The plan score is used to determine when a plan might be removed from the cache. A lower score plan is removed before a higher score plan. By setting the plan score high, the plan remains in the cache for a longer time. Setting the plan score to a low value causes the plan to be pruned sooner than might otherwise have occurred.

Delete allows you to remove the plan immediately from the cache. Note under normal circumstances there might not be a need to modify the attributes of a plan. Normal database processing ages and prunes plans appropriately. These modifying options are provided mostly as tools for minute analysis and for general interest.

The User and Job Name for each statement on the Statements screen is the user and job that created the initial plan with full optimization. This user is not necessarily the same as the last user to run that statement. The Longest Runs screen, however, does show the particular user and job for that individual run.

Filtering Options

The screen provides filtering options which allow the user to more quickly isolate specific criteria of interest. No filters are required to be specified (the default), though adding filtering shortens the time it takes to show the results. The list of statements that is returned is ordered so that the statement consuming the most total processing time is shown at the top. You can reorder the results by clicking the column heading for which you want the list ordered. Repeated clicking toggles the order from ascending to descending.

The filtering options provide a way to focus in on a particular area of interest:

Minimum runtime for the longest execution of the statement:
Show statements with at least one long individual statement instance runtime.

Statements that ran on or after this date and time:
Show statements that have been run recently.

Top 'n' most frequently run statements:
Show statements run most often.

Top 'n' statements with the largest total accumulated runtime:
Show the top resource consumers. Shows the first 'n' top statements by default when no filtering is given. Specifying a value for 'n' improves the performance of getting the first screen of statements, though the total statements displayed is limited to 'n'.

Statements the following user has ever run:
Show statements a particular user has run. The user and job name shown reflect the originator of the cached statement. This user is not necessarily the same as the user specified on the filter (there could be multiple users running the statement).

Statements that are currently active
Show statements that are still running or are in pseudo-close mode. The user and job name shown reflect the originator of the cached statement. This user is not necessarily the same as the user specified on the filter (there could be multiple users running the statement).

Note: An alternative for viewing the active statement for a job is to right-click the Database icon and select **SQL Details for Jobs...**

Statements for which an index has been advised
Show only those statements where the optimizer advised an index to improve performance.

Statements for which statistics have been advised
Show only those statements where a statistic not yet collected might have been useful to the optimizer. The optimizer automatically collects these statistics in the background. This option is normally not that interesting unless, for whatever reason, you want to control the statistics collection yourself.

Include statements initiated by the operating system
Show the 'hidden' statements initiated by the database to process a request. By default the list only includes user-initiated statements.

Statements that reference the following objects:
Show statements that reference the tables or indexes specified.

Statements that contain the following text:
Show statements that include the text specified. This option is useful for finding particular types of statements. For example, statements with a FETCH FIRST clause can be found by specifying ‘fetch'. The search is not case sensitive for ease of use. For example, the string 'FETCH' finds the same statements as the search string 'fetch'. This option provides a wildcard search capability on the SQL text itself.

Multiple filter options can be specified. The candidate statements for each filter are computed independently. Only those statements that are present in all the candidate lists are shown. For example, you could specify options **Top 'n' most frequently run statements** and **Statements the following user has ever run**. The display shows those most frequently run statements in the cache that have been run by the specified user. It does not show the most frequently run statements by the user (unless those statements are also the most frequently run statements in the entire cache).

SQL Plan Cache column descriptions
Displays the columns that are used in the SQL Plan Cache Statements window.

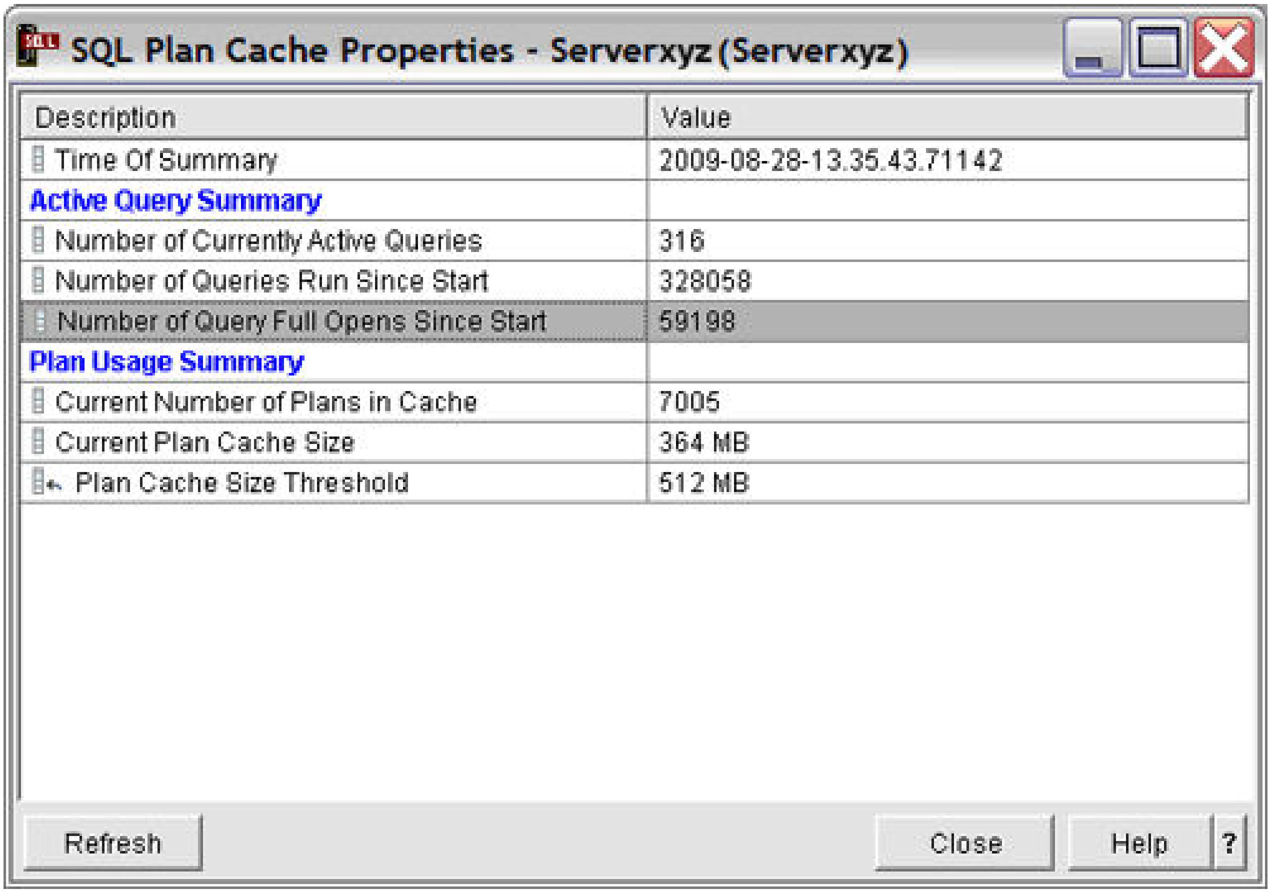
Table 44. Columns used in SQL Plan Cache Statements window

Column name	Description
Last Time Run	Displays the last time that this statement was run.
Most Expensive Time (sec)	The time taken for the longest run of this statement.
Total Processing Time (sec)	The sum total time that all runs of this statement took to process in seconds.
Total Times Run	The total number of times that this statement ran.
Average Processing Time (sec)	The average time per run that this statement took to process in seconds.
Statement	The statement text.
Plan Creation User Name	The name of the user id that created the plan.
Job Name	The name of the job that created the plan.
Job User	The name of the user id that owned the job that created the plan.
Job Number	The job number of the job that created the plan.
Adjusted Average Processing Time (sec)	The average time per run that this statement took to process in seconds where anomalous runs are removed from the average calculation. This time provides a realistic average for a statement by ignoring a single (or few) run that was atypical to the normal condition of the statement.
Average Result Set Rows	The average number of result set rows that are returned when this statement is run.
Average Temp Storage Used (MB)	The average amount of temporary storage used when this statement is run.
Plan Score	The rating of this plan relative to other plans in the cache. A plan with a higher rating relative to other plans remains in the cache for a longer time. A plan with a lower rating relative to other plans is removed from the cache sooner than the other plans.
Plan Identifier	A unique numeric identifier of the plan.
Total Cached Results Used	The number of times a result set from a prior run of the statement was reused on a subsequent run of the statement.
Optimization Time (sec)	The amount of time that it took to optimize this statement.
System Name	The system name.
Relational Database name	Relational database name

SQL plan cache properties
The **SQL Plan Cache** > **Properties** option shows high-level information about the cache. This information includes cache size, number of plans, number of full opens and pseudo-opens that have occurred.

This information can be used to view overall database activity. If tracked over time, it provides trends to help you better understand the database utilization peaks and valleys throughout the day and week.

You can edit the Plan Cache Size Threshold property of your plan cache by right-clicking a property and selecting Edit Value. Under normal circumstances, this properties value is sufficient and altering is not necessary. If it is altered, take care to assess the performance consequences of the change. Note that any explicit designation of Plan Cache size is maintained across IPLs.

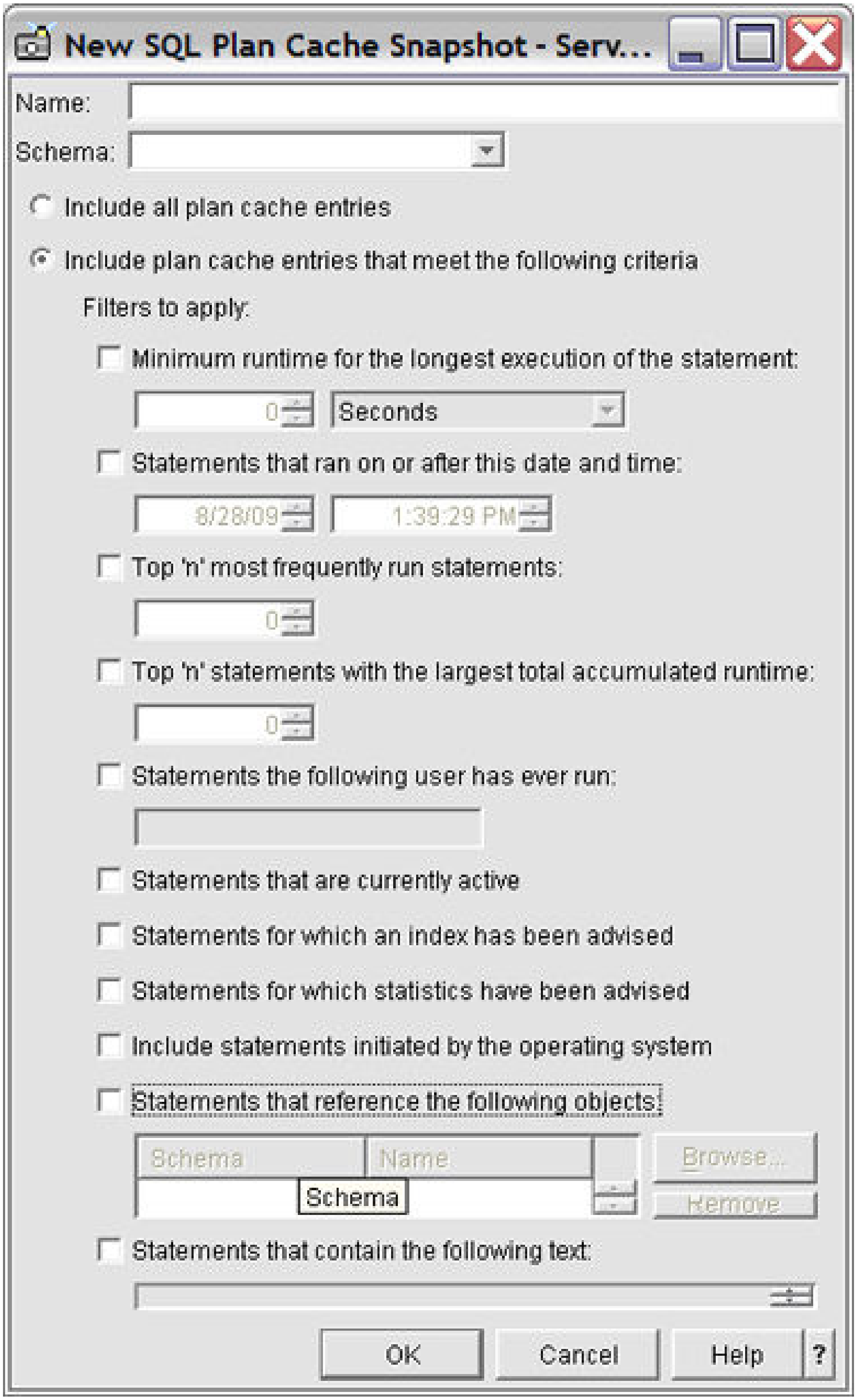


You might be able to edit some of the properties of your plan cache by right-clicking a property and selecting **Edit Value**.

Creating SQL plan cache snapshots

The **New** > **Snapshot** option allows for the creation of a snapshot from the plan cache.

Unlike the snapshot option under Show Statements, it allows you to create a snapshot without having to first view the queries.



The same filtering options are provided here as on the Show Statements screen.

The stored procedure, qsys2.dump_plan_cache, provides the simplest, programmatic way to create a database monitor file output (snapshot) from the plan cache. The dump_plan_cache procedure takes two parameters, library name and file name, for identifying the resulting database monitor file. If the file does not exist, it is created. For example, to dump the plan cache to a database performance monitor file in library QGPL:

```
CALL qsys2.dump_plan_cache('QGPL','SNAPSHOT1');
```

SQL plan cache event monitor
The SQL plan cache event monitor records changes in your plan cache.

You can access the SQL plan cache event monitor through the System i Navigator interface or by calling the procedures directly.

The SQL plan cache event monitor captures monitor records of plans as they are removed from the plan cache. The event monitor is useful for ensuring that all performance information potentially available in the cache is captured even if plans are removed from the cache. Combining the event monitor output with a plan cache snapshot provides a composite view of the cache from when the event monitor was started until the snapshot is taken.

The event monitor allows the same filtering options as described for **Show statements** and **NewSnapshot**. The exceptions are that the *Top 'n' most frequently run statements* and the *Top 'n' statements with largest total accumulated runtime* are not shown. Once a statement is removed from the cache, it is no longer compared to other plans. Therefore, these two 'Top n' filters do not make sense for pruned plans.

Accessing the SQL plan cache with SQL stored procedures
The System i Navigator provides a visual interface into the plan cache. However, the plan cache is also accessible through stored procedures which can be called using the SQL CALL statement.

These procedures allow for programmatic access to the plan cache and can be used, for example, for scheduling plan cache captures or pre-starting an event monitor.

qsys2.dump_plan_cache(‘lib', 'file')

This procedure creates a snapshot (database monitor file) of the contents of the cache. It takes two parameters, library name and file name, for identifying the resulting database monitor file. If the file does not exist, it is created. The file name is restricted to 10 characters.

For example, to dump the plan cache to a database performance monitor file called SNAPSHOT1 in library QGPL:

```
CALL qsys2.dump_plan_cache('QGPL','SNAPSHOT1');
```

qsys2.start_plan_cache_event_monitor(‘lib', 'file')

This procedure starts an event monitor to intercept plans as they are removed from the cache and generate performance information into the specified database monitor file. It takes two parameters, library name and file name, for identifying the resulting database monitor file.

If the file does not exist, it is created. Initially the file is created and populated with the starting record id 3018 (column QQRID = 3018). Control returns to the caller but the event monitor stays active. Library QTEMP is not allowed. The file name is restricted to 10 characters.

The event monitor stays active until one of the following occurs:

- it is ended by one of the end event monitor procedure calls.
- it is ended using the System i Navigator interface.
- an IPL (Initial Program Load) of the system occurs.
- the specified database monitor file is deleted or otherwise becomes unavailable.

For example, to start an event monitor and place plan information into a database performance monitor file called PRUNEDP1 in library QGPL:

```
CALL qsys2.start_plan_cache_event_monitor('QGPL','PRUNEDP1');
```

qsys2.start_plan_cache_event_monitor('lib', 'file', monitorID)

This procedure starts an event monitor to capture plans as they are removed from the cache and generate performance information into a database monitor file. It takes three parameters, library name, file name, and monitorID. The library name and file name identify the resulting database monitor file.

If the file does not exist, it is created. Initially the file is created and populated with the starting record id 3018. The monitorID is a CHAR(10) output parameter set by the database to contain the 10 character identification of the event monitor that was started. Control returns to the procedure caller but the event monitor stays active. Library QTEMP is not allowed. The file name is restricted to 10 characters.

The event monitor stays active until one of the following occurs:

- it is ended by one of the end event monitor procedure calls.
- it is ended using the System i Navigator interface.
- an IPL (Initial Program Load) of the system occurs.
- the specified database monitor file is deleted or otherwise becomes unavailable.

For example, start an event monitor to place plan information into a database performance monitor file called PRUNEDPLANS1 in library QGPL. Capture the monitor id into host variable HVmonid for use later:

```
CALL qsys2.start_plan_cache_event_monitor('QGPL','PRUNEDP1', :HVmonid);
```

qsys2.end_all_plan_cache_event_monitors()

This procedure can be used to end all active plan cache event monitors started either through the GUI or use the start_plan_cache_event_monitor procedures. It takes no parameters.

```
CALL qsys2.end_all_plan_cache_event_monitors();
```

qsys2.end_plan_cache_event_monitor('monID')

This procedure can be used to end the specific event monitor identified by the given monitor id value. This procedure works with the start_plan_plan_event_monitor to end a particular event monitor.

Example:

```
CALL qsys2.end_plan_cache_event_monitor('PLANC00001');
```

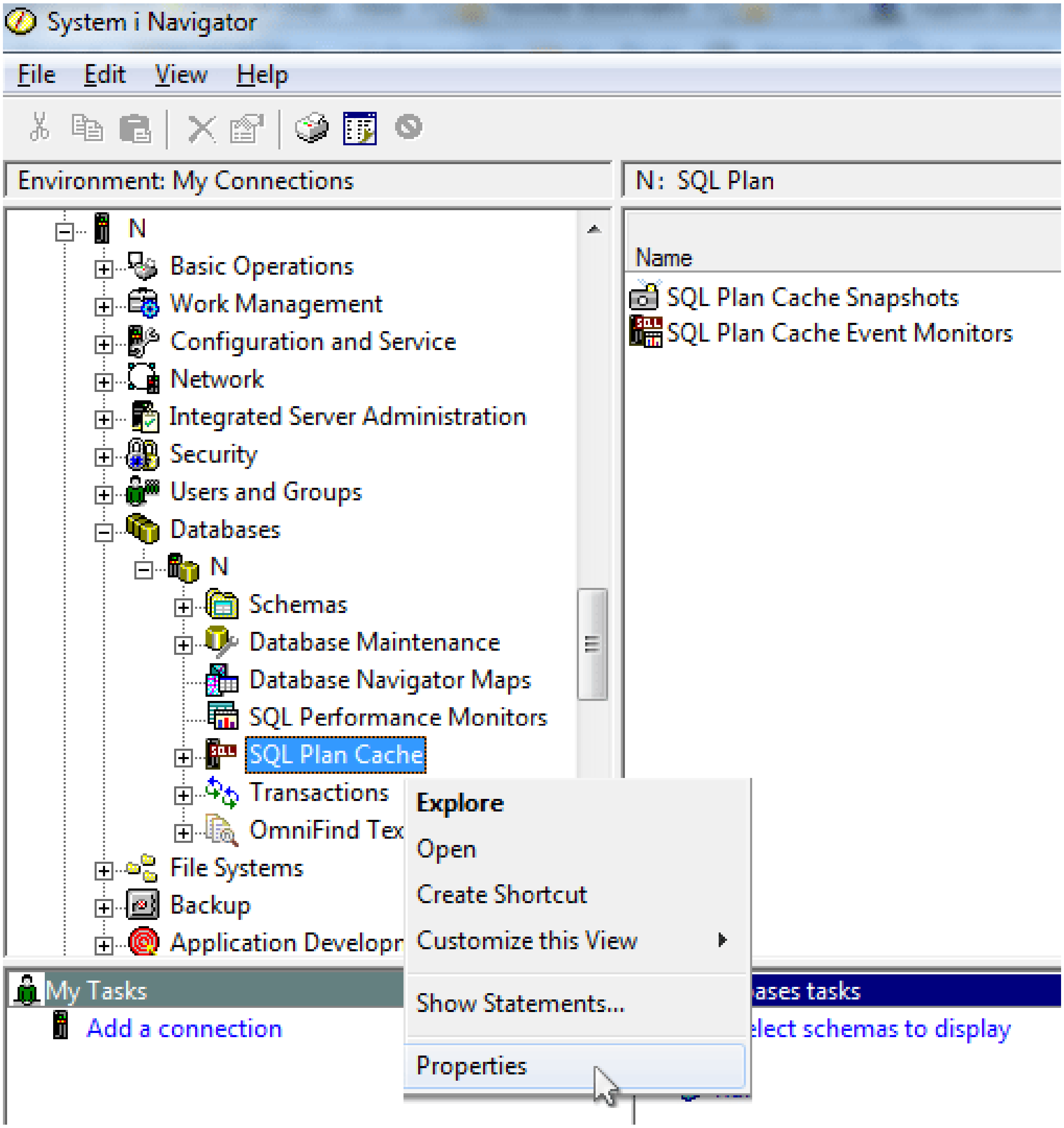
qsys2.change_plan_cache_size(sizeinMeg)

This procedure can be used to change the size of the Plan Cache. The integer parameter specifies the size in megabytes that the plan cache is set to. Once designated, that size will remain at the fixed maximum size of the Plan Cache even across IPLs. If the value given is zero, the plan cache is reset to its default value which allows the plan cache to be auto-sized by the database.

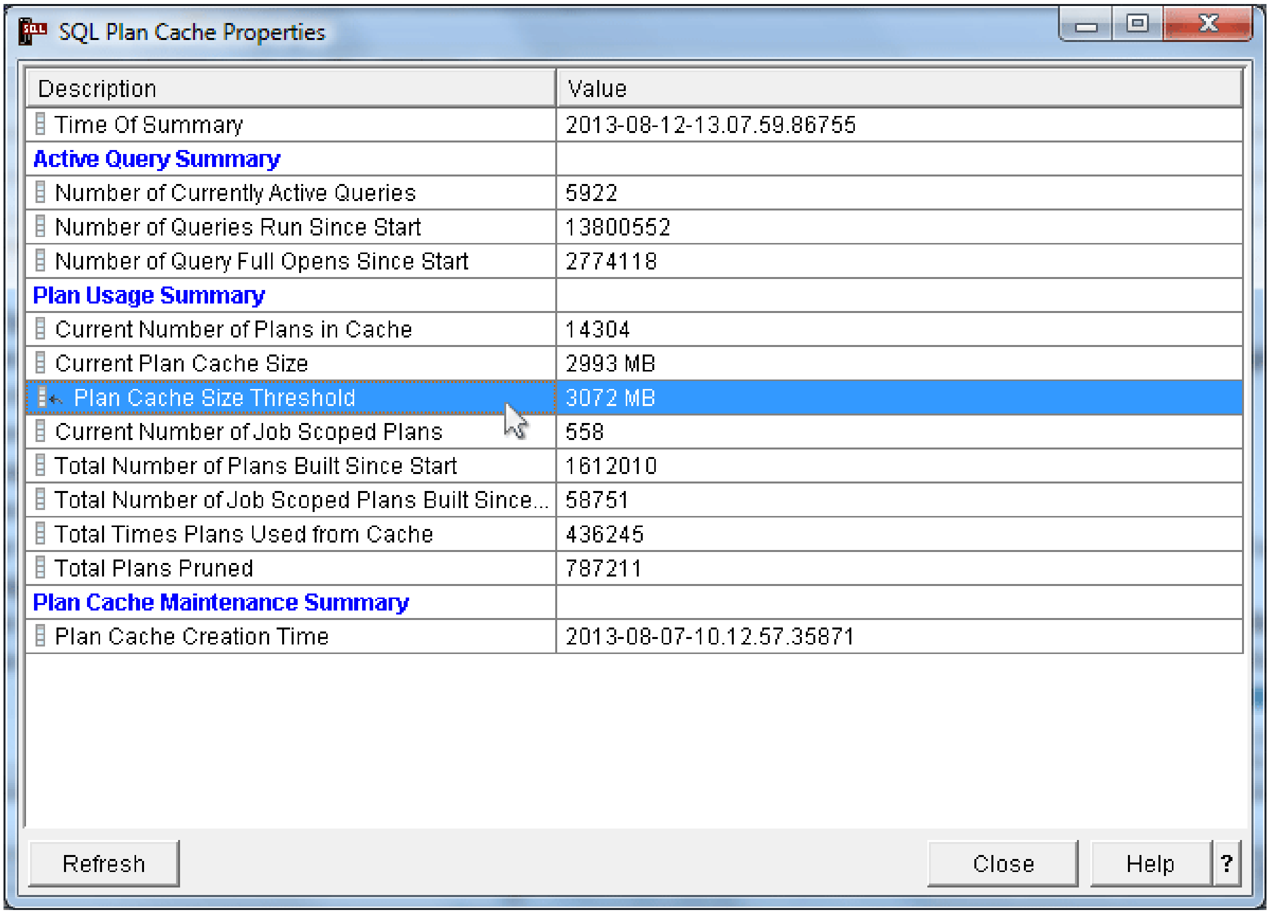
Example:

```
CALL qsys2.change_plan_cache_size(3072);
```

The following graphic illustrates that from System i Navigator, you can select the SQL Plan Cache Properties.



The next graphic shows that the selected SQL Plan Cache properties displays the current plan cache size and the timestamp that it was created.



Description	Value
Time Of Summary	2013-08-12-13.07.59.86755
Active Query Summary	
Number of Currently Active Queries	5922
Number of Queries Run Since Start	13800552
Number of Query Full Opens Since Start	2774118
Plan Usage Summary	
Current Number of Plans in Cache	14304
Current Plan Cache Size	2993 MB
Plan Cache Size Threshold	3072 MB
Current Number of Job Scoped Plans	558
Total Number of Plans Built Since Start	1612010
Total Number of Job Scoped Plans Built Since...	58751
Total Times Plans Used from Cache	436245
Total Plans Pruned	787211
Plan Cache Maintenance Summary	
Plan Cache Creation Time	2013-08-07-10.12.57.35871

qsys2.dump_plan_cache_properties('lib', 'file')

This procedure creates a file containing the properties of the cache. It takes two parameters, library name and file name, for identifying the resulting properties file. If the file does not exist, it is created. The file name is restricted to 10 characters. The file definition matches the archive file qsys2/qdboppcgen.

For example, to dump the plan cache properties to a file called PCPROP1 in library QGPL:

```
CALL qsys2.dump_plan_cache_properties('QGPL','PCPROP1');
```

qsys2.dump_plan_cache_topN('lib', 'file', <number-of-TOP-queries-to-dump>)

This procedure creates a snapshot file from the active plan cache containing only those queries with the largest accumulated elapsed time. The number of queries to capture is designated by the caller in the third parameter. This procedure provides a programmatic way capture the most noteworthy queries, making it easier to compare and contrast this aspect of database performance.

For example, to capture the 20 queries with the largest elapsed time and dump the details into a snapshot file named SNAPSHOTS/TOPN121413:

```
CALL QSYS2.DUMP_PLAN_CACHE_topN('SNAPSHOTS', 'TOPN121413', 20);
```

qsys2.extract_statements

This procedure returns details from a plan cache snapshot in the form of an SQL table or a result set.

The parameters for this function are defined as follows. Only the first 2 parameters are required. The others are optional and will be assumed to be the null value and ignored if they are not specified.

MONITOR_SCHEMA
VARCHAR(10). The schema name for the monitor to use for the extract. This parameter is required.

MONITOR_NAME
VARCHAR(10). The name for the monitor to use for the extract. This parameter is required.

ADDITIONAL_SELECT_COLUMNS
VARCHAR(5000). A character string containing additional columns or expressions to be appended to the generated SELECT clause. A value of *AUDIT will cause the procedure to return the merged statement and columns that are normally interesting to auditing. This parameter is optional.

ADDITIONAL_PREDICATES
VARCHAR(5000). A character string containing additional predicates to be appended to the generated WHERE clause. This parameter is optional.

ORDER_BY
VARCHAR(5000). A character string containing additional options to be appended to the end of the generated query. This can include the ORDER BY clause or other clauses such as FETCH FIRST n ROWS. This parameter is optional.

OUTPUT_SCHEMA
VARCHAR(258). The schema name for the output table. This parameter is optional.

OUTPUT_TABLE
VARCHAR(258). The table name to contain the output. If the table identified by OUTPUT_SCHEMA and OUTPUT_TABLE does not exist, it will be created. If the table exists, the result of this procedure call will be added to the table. This parameter is optional.

If the OUTPUT_SCHEMA and OUTPUT_TABLE parameters have the null value, a result set containing the extracted statement information is returned.

For example, extract the 100 most recent statements from monitor APRIL1014:

```
CALL QSYS2.DUMP_PLAN_CACHE('SNAPSHOTS', 'APRIL2014');

CALL QSYS2.EXTRACT_STATEMENTS('SNAPSHOTS', 'APRIL2014', '*AUDIT',
  'AND QQC21 NOT IN
   (''CH'', ''CL'', ''CN'', ''DE'', ''DI'', ''DM'', ''HC'', ''HH'', ''JR'', ''FE'',
    ''PD'', ''PR'', ''PD'')',
  ' ORDER BY QQSTIM DESC FETCH FIRST 100 ROWS ONLY ');
```

For example, extract all the queries where the query took longer than one second:

```
CALL QSYS2.DUMP_PLAN_CACHE('SNAPSHOTS', 'APRIL2014');

CALL QSYS2.EXTRACT_STATEMENTS('SNAPSHOTS', 'APRIL2014',
ADDITIONAL_SELECT_COLUMNS => ‘DEC(QQI6)/1000000.0 as Total_time,
                              QVC102 as Current_User_Profile ',
ADDITIONAL_PREDICATES => ' AND QQI6 > 1000000 ',
ORDER_BY => ' ORDER BY QQI6 DESC ');
```

qsys2.import_pc_snapshot

This procedure is a programmatic alternative to using System i Navigator to import an existing SQL plan cache snapshot.The caller of this procedure must have the necessary authorities needed to query the target snapshot file.

The parameters for this function are defined as follows. All parameters are required.

PLAN_CACHE_LIBRARY
VARCHAR(10). The library name of the snapshot to import.

PLAN_CACHE_FILE
VARCHAR(10). The file name of the snapshot to import.

IMPORTED_NAME
CHAR(30). The character string that describes the snapshot being imported. This string will appear in Navigator's Name column under SQL Plan Cache Snapshots.

This example shows how to programmatically capture information from the live plan cache for the 50 most expensive queries and import the snapshot into Navigator.

```
CALL QSYS2.DUMP_PLAN_CACHE_TOPN('SNAPSHOTS', 'JUNE2014', 50);
CALL QSYS2.IMPORT_PC_SNAPSHOT('SNAPSHOTS', 'JUNE2014', 'Top 50 Queries-June 2014');
```

qsys2.remove_pc_snapshot

This procedure is a programmatic alternative to using System i Navigator to delete an existing SQL plan cache snapshot. The caller of this procedure must have the necessary authorities needed to delete the target file. Any rows in the Navigator snapshot list which correspond to the input library and file name are removed. The input file name is deleted.

The parameters for this function are defined as follows. All parameters are required.

PLAN_CACHE_LIBRARY
VARCHAR(10). The library name of the snapshot to remove.

PLAN_CACHE_FILE
VARCHAR(10). The file name of the snapshot to remove.

This example shows how to programmatically remove a snapshot that has aged beyond its usefulness.

```
CALL QSYS2.REMOVE_PC_SNAPSHOT('SNAPSHOTS', 'JUNE2013');
```

qsys2.import_pc_event_monitor

This procedure is a programmatic alternative to using System i Navigator to import an existing SQL plan cache event monitor. The caller of this procedure must have the necessary authorities needed to query the target event monitor file.

The parameters for this function are defined as follows. All parameters are required.

PLAN_CACHE_LIBRARY
VARCHAR(10). The library name of the event monitor to import.

PLAN_CACHE_FILE
VARCHAR(10). The file name of the event monitor to import.

IMPORTED_NAME
CHAR(30). The character string that describes the event monitor being imported. This string will appear in Navigator's Name column under SQL Plan Cache Event Monitors.

This example shows how to programmatically import an event monitor into Navigator that was used to capture queries pruned from the plan cache during the month of June, 2014.

```
CALL QSYS2.IMPORT_PC_EVENT_MONITOR('SNAPSHOTS', 'PRUNE0614', 'Pruned queries - June 2014');
```

qsys2.remove_pc_event_monitor

This procedure is a programmatic alternative to using System i Navigator to delete an existing SQL plan cache event monitor. The caller of this procedure must have the necessary authorities needed to delete the target file. Any rows in the Navigator event monitor list which correspond to the input library and file name are removed. The input file name is deleted.

The parameters for this function are defined as follows. All parameters are required.

PLAN_CACHE_LIBRARY
VARCHAR(10). The library name of the event monitor to remove.

PLAN_CACHE_FILE
VARCHAR(10). The file name of the event monitor to remove.

This example shows how to programmatically remove an event monitor that has aged beyond its usefulness.

```
CALL QSYS2.REMOVE_PC_EVENT_MONITOR('SNAPSHOTS', 'PRUNE0613');
```

CLEAR_PLAN_CACHE
The **CLEAR_PLAN_CACHE** procedure is a plan cache clearing alternative to performing a system IPL.

CLEAR_PLAN_CACHE procedure

```
  >>-CLEAR_PLAN_CACHE --()---------------------------------><
```

The schema is QSYS2.

This procedure is used primarily in performance test and QA environments. It provides database performance analysts with a way to create a consistent environment from which to evaluate potential database performance changes. The procedure will clear all plans in the system SQL Plan Cache that exist at the time the procedure is run. Besides clearing the plan information, any Maintained Temporary Indexes (MTIs) not currently in use by a query will be deleted as part of the clear. SQL queries run while the **CLEAR_PLAN_CACHE** procedure is running may have their plans removed, but the queries themselves will not incur a failure related to plan removal. Any SQL queries run after the clear is complete will begin to repopulate the plan cache.

The time the **CLEAR_PLAN_CACHE** procedure takes to run will vary depending on the plan cache size. To avoid tying up an interactive job, it is recommended to submit the procedure in a background job using a combination of the Submit Job (**SBMJOB**) and Run SQL (**RUNSQL**) CL commands.

Authorization: The **CLEAR_PLAN_CACHE** procedure requires that the authorization ID associated with the statement has *JOBCTL special authority or QIBM_DB_SQLADM function usage.

Errors: The procedure will fail with SQL0443 and SQL0552 if the caller does not have the required authority.

Example:

```
SBMJOB CMD(RUNSQL SQL('CALL QSYS2.CLEAR_PLAN_CACHE()') COMMIT(*NONE) NAMING(*SQL))
```

Verifying the performance of SQL applications

You can verify the performance of an SQL application by using commands.

The commands that can help you verify performance:

Display Job (DSPJOB)
You can use the **Display Job (DSPJOB)** command with the OPTION(*OPNF) parameter to show the indexes and tables used by an application running in a job.

You can also use **DSPJOB** with the OPTION(*JOBLCK) parameter to analyze object and row lock contention. It displays the objects and rows that are locked and the name of the job holding the lock.

Specify the OPTION(*CMTCTL) parameter on the **DSPJOB** command to show the isolation level, the number of rows locked during a transaction, and the pending DDL functions. The isolation level displayed is the default isolation level. The actual isolation level, used for any SQL program, is specified on the COMMIT parameter of the CRTSQLxxx command.

Print SQL Information (PRTSQLINF)
The **Print SQL Information (PRTSQLINF)** command lets you print information about the embedded SQL statements in a program, SQL package, or service program. The information includes the SQL statements, access plans used, and the command parameters used to precompile the source member.

Start Database Monitor (STRDBMON)
You can use the **Start Database Monitor (STRDBMON)** command to capture to a file information about every SQL statement that runs.

Change Query Attribute (CHGQRYA)
: You can use the **Change Query Attribute (CHGQRYA)** command to change the query attributes for the query optimizer. Among the attributes that can be changed by this command are the predictive query governor, parallelism, and the query options.

Start Debug (STRDBG)
: You can use the **Start Debug (STRDBG)** command to put a job into debug mode, and optionally add as many as 20 programs, 20 class files, and 20 service programs to debug mode. It also specifies certain attributes of the debugging session. For example, it can specify whether database files in production libraries can be updated while in debug mode.

Related information

Display Job (DSPJOB) command
Print SQL Information (PRTSQLINF) command
Start Database Monitor (STRDBMON) command
Change Query Attributes (CHGQRYA) command
Start Debug (STRDBG) command

Examining query optimizer debug messages in the job log

Query optimizer debug messages issue informational messages to the job log about the implementation of a query. These messages explain what happened during the query optimization process.

For example, you can learn:

- Why an index was or was not used
- Why a temporary result was required
- Whether joins and blocking are used
- What type of index was advised by the optimizer
- Status of the job queries
- Indexes used
- Status of the cursor

The optimizer automatically logs messages for all queries it optimizes, including SQL, call level interface, ODBC, OPNQRYF, and SQL Query Manager.

Viewing debug messages using STRDBG command:

STRDBG command puts a job into debug mode. It also specifies certain attributes of the debugging session. For example, it can specify whether database files in production schemas can be updated while in debug mode. For example, use the following command:

```
STRDBG PGM(Schema/program) UPDPROD(*YES)
```

STRDBG places in the job log information about all SQL statements that run.

Viewing debug messages using QAQQINI table:

You can also set the QRYOPTLIB parameter on the **Change Query Attributes (CHGQRYA)** command to a user schema where the QAQQINI table exists. Set the parameter on the QAQQINI table to MESSAGES_DEBUG, and set the value to *YES. This option places query optimization information in the job log. Changes made to the QAQQINI table are effective immediately and affect all users and queries that use this table. Once you change the MESSAGES_DEBUG parameter, all queries that use this QAQQINI table write debug messages to their respective job logs. Pressing F10 from the command Entry panel displays the message text. To see the second-level text, press F1 (Help). The second-level text sometimes offers hints for improving query performance.

Viewing debug messages in Run SQL Scripts:

To view debug messages in Run SQL Scripts, from the **Options** menu, select **Include Debug Messages in Job Log**. Then from the **View** menu, select **Job Log**. To view detailed messages, double-click a message.

Viewing debug messages in Visual Explain:

In Visual Explain, debug messages are always available. You do not need to turn them on or off. Debug messages appear in the lower portion of the window. You can view detailed messages by double-clicking a message.

Print SQL Information

The **Print SQL Information (PRTSQLINF)** command returns information about the embedded SQL statements in a program, SQL package (used to store the access plan for a remote query), or service program. This information is then stored in a spooled file.

PRTSQLINF provides information about:

- The SQL statements being executed
- The type of access plan used during execution. How the query is implemented, indexes used, join order, whether a sort is used, whether a database scan is used, and whether an index is created.
- A list of the command parameters used to precompile the source member for the object.
- The CREATE PROCEDURE and CREATE FUNCTION statement text used to create external procedures or User Defined Functions.

This output is like the information that you can get from debug messages. However, while query debug messages work at runtime, **PRTSQLINF** works retroactively. You can also see this information in the second-level text of the query governor inquiry message CPA4259.

You can issue **PRTSQLINF** in a couple of ways. First, you can run the **PRTSQLINF** command against a saved access plan. You must execute or at least prepare the query (using the SQL PREPARE statement) before you use the command. It is best to execute the query because the index created as a result of PREPARE is relatively sparse. It could well change after the first run. **PRTSQLINF**'s requirement of a saved access plan means that the command cannot be used with **OPNQRYF**.

You can also run **PRTSQLINF** against functions, stored procedures, triggers, SQL packages, and programs from System i Navigator. This function is called Explain SQL. To view **PRTSQLINF** information, right-click an object and select **Explain SQL**.

Related information

Print SQL Information (PRTSQLINF) command

Query optimization tools: Comparison

Use this table to find the information each tool can provide, when it analyzes your queries, and the tasks it can do to improve your queries.

Table 45. Optimization tool comparison

PRTSQLINF	STRDBG or CHGQRYA	File-based monitor (STRDBMON)	Memory-Based Monitor	Visual Explain
Available without running query (after access plan has been created)	Only available when the query is run	Only available when the query is run	Only available when the query is run	Only available when the query is explained
Displayed for all queries in SQL program, whether executed or not	Displayed only for those queries which are executed	Displayed only for those queries which are executed	Displayed only for those queries which are executed	Displayed only for those queries that are explained

Table 45. Optimization tool comparison (continued)				
PRTSQLINF	**STRDBG or CHGQRYA**	**File-based monitor (STRDBMON)**	**Memory-Based Monitor**	**Visual Explain**
Information about host variable implementation	Limited information about the implementation of host variables	All information about host variables, implementation, and values	All information about host variables, implementation, and values	All information about host variables, implementation, and values
Available only to SQL users with programs, packages, or service programs	Available to all query users (**OPNQRYF**, SQL, QUERY/400)	Available to all query users (**OPNQRYF**, SQL, QUERY/400)	Available only to SQL interfaces	Available through System i Navigator Database and API interface
Messages are printed to spool file	Messages are displayed in job log	Performance rows are written to database table	Performance information is collected in memory and then written to database table	Information is displayed visually through System i Navigator
Easier to tie messages to query with subqueries or unions	Difficult to tie messages to query with subqueries or unions	Uniquely identifies every query, subquery, and materialized view	Repeated query requests are summarized	Easy to view implementation of the query and associated information

Changing the attributes of your queries

You can modify different types of query attributes for a job with the **Change Query Attributes (CHGQRYA)** CL command. You can also use the System i Navigator Change Query Attributes interface.

Related concepts

Plan cache
The plan cache is a repository that contains the access plans for queries that were optimized by SQE.

Objects processed in parallel
The DB2 Symmetric multiprocessing feature provides the optimizer with additional methods for retrieving data that include parallel processing. Symmetrical multiprocessing is a form of parallelism achieved on a single system where multiple CPU and I/O processors sharing memory and disk work simultaneously toward a single result.

Related information

Change Query Attributes (CHGQRYA) command

Controlling queries dynamically with the query options file QAQQINI

The query options file QAQQINI support provides the ability to dynamically modify or override the environment in which queries are executed. This modification is done through the **Change Query Attributes (CHGQRYA)** command and the QAQQINI file. The query options file QAQQINI is used to set some attributes used by the database manager.

For each query that is run the query option values are retrieved from the QAQQINI file in the schema specified on the QRYOPTLIB parameter of the CHGQRYA CL command and used to optimize or implement the query.

Environmental attributes that you can modify through the QAQQINI file include:

- ALLOW_ADAPTIVE_QUERY_PROCESSING
- ALLOW_ARRAY_VALUE_CHANGES
- ALLOW_TEMPORARY_INDEXES
- APPLY_REMOTE
- ASYNC_JOB_USAGE

- CACHE_RESULTS
- COLLATE_ERRORS
- COMMITMENT_CONTROL_LOCK_LIMIT
- DETERMINISTIC_UDF_SCOPE
- FIELDPROC_ENCODED_COMPARISON
- FORCE_JOIN_ORDER
- IGNORE_LIKE_REDUNDANT_SHIFTS
- LIMIT_PREDICATE_ OPTIMIZATION
- LOB_LOCATOR_THRESHOLD
- MATERIALIZED_QUERY_TABLE_REFRESH_AGE
- MATERIALIZED_QUERY_TABLE _USAGE
- MEMORY_POOL_PREFERENCE
- MESSAGES_DEBUG
- NORMALIZE_DATA
- OPEN_CURSOR_CLOSE_COUNT
- OPEN_CURSOR_THRESHOLD
- OPTIMIZATION_GOAL
- OPTIMIZE_STATISTIC_LIMITATION
- PARALLEL_DEGREE
- PARAMETER_MARKER_CONVERSION
- PSEUDO_OPEN_CHECK_HOST_VARS
- QUERY_TIME_LIMIT
- REOPTIMIZE_ACCESS_PLAN
- SQLSTANDARDS_MIXED_CONSTANT
- SQL_CONCURRENT_ACCESS_RESOLUTION
- SQL_DECFLOAT_WARNINGS
- SQL_FAST_DELETE_ROW_COUNT
- SQL_GVAR_BUILD_RULE
- SQL_MODIFIES_SQL_DATA
- SQL_PSEUDO_CLOSE
- SQL_STMT_COMPRESS_MAX
- SQL_STMT_REUSE
- SQL_SUPPRESS_WARNINGS
- SQL_TRANSLATE_ASCII_TO_JOB
- SQL_XML_DATA_CCSID
- STAR_JOIN
- STORAGE_LIMIT
- SYSTEM_SQL_STATEMENT_CACHE
- TEXT_SEARCH_DEFAULT_TIMEZONE
- UDF_TIME_OUT
- VARIABLE_LENGTH_OPTIMIZATION

Specifying the QAQQINI file with CHGQRYA

Use the **Change Query Attributes (CHGQRYA)** command with the QRYOPTLIB (query options library) parameter to specify which schema currently contains or contains the query options file QAQQINI.

The query options file is retrieved from the schema specified on the QRYOPTLIB parameter for each query. It remains in effect for the duration of the job or user session, or until the QRYOPTLIB parameter is changed by the **Change Query Attributes (CHGQRYA)** command.

If the **Change Query Attributes (CHGQRYA)** command is not issued, or is issued without the QRYOPTLIB parameter specified, QUSRSYS is searched for the QAQQINI file. If a query options file is not found, no attributes are modified. Since the system ships without an INI file in QUSRSYS, you might receive a message indicating that there is no INI file. This message is not an error but an indication that a QAQQINI file that contains all default values is being used. The initial value of the QRYOPTLIB parameter for a job is QUSRSYS.

Related information

Change Query Attributes (CHGQRYA) command

Creating the QAQQINI query options file

Each system is shipped with a QAQQINI template file in schema QSYS. The QAQQINI file in QSYS is to be used as a template when creating all user specified QAQQINI files.

To create your own QAQQINI file, use the **Create Duplicate Object (CRTDUPOBJ)** command. Create a copy of the QAQQINI file in the schema specified on the **Change Query Attributes (CHGQRYA)** QRYOPTLIB parameter. The file name must remain QAQQINI. For example:

```
CRTDUPOBJ OBJ(QAQQINI)
          FROMLIB(QSYS)
          OBJTYPE(*FILE)
          TOLIB(MYLIB)
          DATA(*YES)
```

System-supplied triggers are attached to the QAQQINI file in QSYS therefore it is imperative that the only means of copying the QAQQINI file is through the CRTDUPOBJ CL command. If another means is used, such as **CPYF**, then the triggers could be corrupted. An error is signaled that the options file cannot be retrieved or that the options file cannot be updated.

Because of the trigger programs attached to the QAQQINI file, the following CPI321A informational message is displayed six times in the job log when the **CRTDUPOBJ** CL is used to create the file. These messages are not an error; they are only informational messages.

CPI321A Information Message: Trigger QSYS_TRIG_&1___QAQQINI___00000&N in library &1 was added to file QAQQINI in library &1. The ampersand variables (&1, &N) are replacement variables that contain either the library name or a numeric value.

Note: It is highly recommended that the file QAQQINI, in QSYS, not be modified. This file is the original template that is duplicated into QUSRSYS or a user specified library for use.

Related information

Change Query Attributes (CHGQRYA) command

Create Duplicate Object (CRTDUPOBJ) command

QAQQINI file override support

If you find working with the QAQQINI query options file cumbersome, consider using the QSYS2.OVERRIDE_QAQQINI procedure. Instead of creating, managing, and using a QAQQINI *FILE object directly, this procedure can be called to work with a temporary version of the INI file. It uses user-specified options and values. The support relies upon the QTEMP library, so any changes affect only the job which calls the procedure.

See OVERRIDE_QAQQINI procedure for more information.

QAQQINI query options file format

The QAQQINI file is shipped in the schema QSYS. It has a predefined format and has been pre-populated with the default values for the rows.

Query Options File:

```
A                                                    UNIQUE
A                    R QAQQINI                       TEXT('Query options + file')
A                      QQPARM      256A              VARLEN(10) +
                                                     TEXT('Query+
                                                           option parameter') +
                                                     COLHDG('Parameter')
A                      QQVAL       256A              VARLEN(10) +
                                                     TEXT('Query option +
                                                           parameter value') +
                                                     COLHDG('Parameter Value')
A                      QQTEXT      1000G             VARLEN(100) +
                                                     TEXT('Query +
                                                           option text') +
                                                     ALWNULL +
                                                     COLHDG('Query Option' +
                                                             'Text') +
                                                     CCSID(13488) +
                                                     DFT(*NULL)
A                    K QQPARM
```

Setting the options within the query options file

The QAQQINI file query options can be modified with the INSERT, UPDATE, or DELETE SQL statements.

For the following examples, a QAQQINI file has already been created in library MyLib. To update an existing row in MyLib/QAQQINI use the UPDATE SQL statement. This example sets MESSAGES_DEBUG = *YES so that the query optimizer prints out the optimizer debug messages:

```
UPDATE MyLib/QAQQINI SET QQVAL='*YES'
WHERE QQPARM='MESSAGES_DEBUG'
```

To delete an existing row in MyLib/QAQQINI use the DELETE SQL statement. This example removes the QUERY_TIME_LIMIT row from the QAQQINI file:

```
DELETE FROM MyLib/QAQQINI
WHERE QQPARM='QUERY_TIME_LIMIT'
```

To insert a new row into MyLib/QAQQINI use the INSERT SQL statement. This example adds the QUERY_TIME_LIMIT row with a value of *NOMAX to the QAQQINI file:

```
 INSERT INTO MyLib/QAQQINI
VALUES('QUERY_TIME_LIMIT','*NOMAX','New time limit set by DBAdmin')
```

QAQQINI query options file authority requirements

QAQQINI is shipped with a *PUBLIC *USE authority. This authority allows users to view the query options file, but not change it. Changing the values of the QAQQINI file affects all queries run on the system. Allow only the system or database administrator to have *CHANGE authority to the QAQQINI query options file.

The query options file, which resides in the library specified on the **Change Query Attributes (CHGQRYA)** CL command QRYOPTLIB parameter, is always used by the query optimizer. It is used even if the user has no authority to the query options library and file. This authority provides the system administrator with an additional security mechanism.

When the QAQQINI file resides in the library QUSRSYS the query options affects all the query users on the system. To prevent anyone from inserting, deleting, or updating the query options, the system administrator must remove update authority from *PUBLIC to the file. This update authority prevents users from changing the data in the file.

A copy of the QAQQINI file can also reside in a user library. If that library is specified on the QRYOPTLIB parameter of the **Change Query Attributes (CHGQRYA)** command, the query options affect all the queries run for that user job. To prevent the query options from being retrieved from a particular library the system administrator can revoke authority to the **Change Query Attributes (CHGQRYA)** CL command.

QAQQINI file system-supplied triggers

The query options file QAQQINI file uses a system-supplied trigger program in order to process any changes made to the file. A trigger cannot be removed from or added to the file QAQQINI.

If an error occurs on the update of the QAQQINI file (an INSERT, DELETE, or UPDATE operation), the following SQL0443 diagnostic message is issued:

```
Trigger program or external routine detected an error.
```

QAQQINI query options

There are different options available for parameters in the QAQQINI file.

The following table summarizes the query options that can be specified on the QAQQINI command:

Table 46. Query Options Specified on QAQQINI Command

<table>
<tr><th>Parameter</th><th>Value</th><th>Description</th></tr>
<tr><td rowspan="3">ALLOW_ADAPTIVE_QUERY_PROCESSING

Specifies whether Adaptive Query Processing (AQP) processing is done for a query.

Adaptive query processing uses runtime statistics to look for poor performing queries and potentially replace the poor plan with an improved plan.</td><td>*DEFAULT</td><td>The default value is set to *YES.</td></tr>
<tr><td>*YES</td><td>Allows Adaptive query processing to occur for this query.

The existing QAQQINI options that affect AQP are the following:

• If the REOPTIMIZE_ACCESS_PLAN QAQQINI option is set to *ONLY_REQUIRED, AQP does not reoptimize the original plan. *ONLY_REQUIRED indicates the user does not want the query reoptimized unless there is a functional reason to do so. *ONLY_REQUIRED takes precedence over AQP.
• Join order requirements specified by the user in the FORCE_JOIN_ORDER QAQQINI option take precedence over AQP. If the user specifies the primary table in the join order, any AQP primary recommendations will be placed after the primary table if they are different.</td></tr>
<tr><td>*NO</td><td>Adaptive query processing cannot be used for this query.</td></tr>
</table>

Table 46. Query Options Specified on QAQQINI Command (continued)

Parameter	Value	Description
ALLOW_ARRAY_VALUE_CHANGES Specifies whether changes to the values of array elements are visible to the query while the query is running.	*DEFAULT	The default value is set to *NO.
	*NO	Do not allow changes to values in arrays referenced in the query to be visible after the query is opened. All values which could be referenced in a query are copied during query open processing. Any changes to values in arrays after the query is opened are not visible. Produces queries with predictable and reproducible results, but might have a performance penalty when working with large arrays or large array elements. The penalty is less if all the references to arrays are simple non-column values, for example, `:ARRAY[1]` or `:ARRAY[:hv2]`. Use of column values from a table to index the ARRAY, or using the UNNEST() function results in copies of the entire array being made. These copies have the largest performance penalty.
	*YES	Allow changes to values in arrays to be visible to the query while the query is running. The arrays are not copied during the open processing of the query. If the array values are changed during the processing of queries, the results of the query might be unpredictable. Performance might be improved for queries which reference large arrays in complex array index lookup operations, such as `:Array[column-name]`, or when using UNNEST. Large arrays include arrays that have thousands of elements, or elements with a large size. Array index lookups using simple index values, such as `:ARRAY[1]` or `:ARRAY[:hv2]`, see minimal performance improvements. Performance of some queries might be negatively impacted. For example, later queries that could reuse the results if they were cached to avoid recalculation where the cached result is applicable. Procedures that can run with *YES and still expect predictable results have the following characteristics: 1. Contain no cursor declarations. 2. Receive arrays as input parameters: • and do not contain SET statements which reference arrays on the left side of the SET, and • and have no SQL statements with INTO clauses referencing arrays. 3. Do not contain SET statements which reference arrays on the left side of the set: • and have no SQL statements with INTO clauses referencing arrays while a cursor is open for a query which references an array.
ALLOW_TEMPORARY_ INDEXES Specifies whether temporary indexes can be considered by the optimizer. If temporary indexes are not allowed, then any other viable plan is chosen regardless of cost to implement this query.	*DEFAULT	The default value is set to *YES.
	*YES	Allow temporary indexes to be considered.
	*ONLY_ REQUIRED	Do not allow any temporary indexes to be considered for this access plan. Choose any other implementation regardless of cost to avoid the creation of a temporary index. Only if no viable plan can be found, is a temporary index allowed.
APPLY_REMOTE Specifies for database queries involving distributed files, whether the CHGQRYA query attributes are applied to the jobs on the remote systems associated with this job.	*DEFAULT	The default value is set to *YES.
	*NO	The **CHGQRYA** attributes for the job are not applied to the remote jobs. The remote jobs use the attributes associated to them on their systems.
	*YES	The query attributes for the job are applied to the remote jobs used in processing database queries involving distributed tables. For attributes where *SYSVAL is specified, the system value on the remote system is used for the remote job. This option requires that, if **CHGQRYA** was used for this job, the remote jobs must have authority to use the **CHGQRYA** command.

Table 46. Query Options Specified on QAQQINI Command (continued)

Parameter	Value	Description
ASYNC_JOB_USAGE Specifies the circumstances in which asynchronous (temp writer) jobs can be used to help process database queries in the job. The option determines which types of database queries can be used in asynchronous jobs (running in parallel) to help complete the query. An asynchronous job is a separate job that handles query requests from jobs running the database queries on the system. The asynchronous job processes each request and puts the results into a temporary file. This intermediate temporary file is then used by the main job to complete the database query. The advantage of an asynchronous job is that it processes its request at the same time (in parallel) that the main job processes another query step. The disadvantage of using an asynchronous job is that it might encounter a situation that it cannot handle in the same way as the main job. For example, the asynchronous job might receive an inquiry message from which it cancels, whereas the main job can choose to ignore the message and continue. There are two different types of database queries that can run asynchronous jobs: 1. Distributed queries. These are database queries that involve distributed files. Distributed files are provided through the system feature DB2 Multi-System for IBM i. 2. Local queries. there are database queries that involve only files local to the system where the database queries are being run.	*DEFAULT	The default value is set to *LOCAL.
	*LOCAL	Asynchronous jobs might be used for database queries that involve only tables local to the system where the database queries are being run. In addition, this option allows the communications required for queries involving distributed tables to be asynchronous. Each system involved in the query of the distributed tables can run its portion of the query at the same time (in parallel).
	*DIST	Asynchronous jobs might be used for database queries that involve distributed tables.
	*ANY	Asynchronous jobs might be used for any database query.
	*NONE	No asynchronous jobs are allowed to be used for database query processing. In addition, all processing for queries involving distributed tables occurs synchronously. Therefore, no intersystem parallel processing occurs.
CACHE_RESULTS Specifies a way for the user to control the cache results processing. For queries involving temporary results, for example, sorts or hashes, the database manager often saves the results across query pseudo-close or pseudo-open. The results are saved as long as they are not large, with the hope that they can be reused for the next run of the query. Beginning in V5R3, the database manager saves these temporary results even when a job is finished with them. The database manager assumes that another job can later reuse the results. The database manager automatically controls the caching of these results, removing cache results as storage usage becomes large. However, the amount of temporary storage used by the database can be noticeably more than in previous releases.	*DEFAULT	The default value is the same as *SYSTEM.
	*SYSTEM	The database manager might cache a query result set. A subsequent run of the query by the same job can reuse the cached result set. Or, if the ODP for the query has been deleted, any job can reuse the cached result set.
	*JOB	The database manager might cache a query result set from one run to the next for a job. Caching can occur as long as the query uses a reusable ODP. When the reusable ODP is deleted, the cached result set is destroyed. This value mimics V5R2 processing.
	*NONE	The database does not cache any query results.
COLLATE_ERRORS Specifies how data errors are handled on the GROUP BY and ORDER BY expression during hash or sort processing within queries.	*DEFAULT	The default value is *NO.
	*NO	A value of *NO causes the query to be ended with an error when a grouping or ordering expressions results in an error.
	*YES	A value of *YES indicates that the grouping or sort continues.
COMMITMENT_CONTROL_ LOCK_LIMIT Specifies the maximum number of records that can be locked to a commit transaction initiated after setting the new value. The value specified for COMMITMENT_CONTROL _LOCK_LIMIT does not affect transactions running in jobs that have already started commitment control. For the value to be effective, it must be changed before starting commitment control.	*DEFAULT	*DEFAULT is equivalent to 500,000,000. If multiple journals are involved in the transaction, the COMMITMENT_CONTROL _LOCK_LIMIT applies to each journal, not to the transaction as a whole. For example, files F1 to F5 are journaled to journal J1, and files F6 to F10 are journaled to J2. The COMMITMENT_CONTROL _LOCK_LIMIT is set to 100,000. 100,000 record locks can be acquired for files F1 to F5. 100,000 more locks can be acquired for files F6 to F10.
	Integer Value	The maximum number of records that can be locked to a commit transaction initiated after setting the new value. The valid integer value is 1–500,000,000.

Table 46. Query Options Specified on QAQQINI Command (continued)		
Parameter	**Value**	**Description**
DETERMINISTIC_UDF_SCOPE Specifies the scope or lifetime of the deterministic setting for User Defined Functions (UDFs) and User Defined Table Functions (UDTFs). It is recommended that you specify STATEMENT DETERMINISTIC on any CREATE FUNCTION statement that should be considered deterministic for a single instance of a query open rather than using the *OPEN option. DETERMINISTIC_UDF_SCOPE applies to all deterministic UDFs and UDTFs in every query while this QAQQINI option is in effect.	*DEFAULT	The default value is *ALWAYS.
	*ALWAYS	The UDF is always considered deterministic. Query temporary objects might be shared across query opens and the UDF might not run for a particular query open.
	*OPEN	The UDF is considered deterministic only for a single instance of a query open. Query temporary objects are not shared across query open. The UDF is run at least once in the query for a given set of input parameters.
FIELDPROC_ENCODED_COMPARISON Specifies the amount of optimization that the optimizer might use when queried columns have attached field procedures	*DEFAULT	The default value is *ALLOW_EQUAL.
	*NONE	No optimization to remove field procedure decode option 4 or transformations to optimize field procedure invocations is allowed. For example, the optimizer cannot transform `fieldProc(4, column) = 'literal'` to `column = fieldProc(0, 'literal')`. This option is used when the field procedure is not deterministic.
	*ALLOW_ EQUAL	Optimization allowed for equal and not equal predicates, GROUP BY, and DISTINCT processing. For example, the optimizer might choose to change the predicate `fieldProc(4, column) = 'literal'` to `column = fieldProc(0, 'literal')` in order to facilitate index matching. This option is useful when the field procedure is deterministic but no ordering can be determined based on the result of the field encoding.
	*ALLOW_ RANGE	Transformation allowed for MIN, MAX grouping functions, ORDER BY, and all predicates except LIKE in addition to the transformations supported by *ALLOW_EQUAL. This option is useful when the field procedure is deterministic and the encoded value implies ordering
	*ALL	Transformation allowed for all predicates including LIKE, in addition to the transformations supported by *ALLOW_RANGE.
FORCE_JOIN_ORDER Specifies to the query optimizer that the join of files is to occur in the order specified in the query.	*DEFAULT	The default is set to *NO.
	*NO	Allow the optimizer to reorder join tables.
	*SQL	Only force the join order for those queries that use the SQL JOIN syntax. This option mimics the behavior for the optimizer before V4R4M0.
	*PRIMARY nnn	Only force the join position for the file listed by the numeric value nnn into the primary position (or dial) for the join. nnn is optional and defaults to 1. The optimizer then determines the join order for all the remaining files based upon cost.
	*YES	Do not allow the query optimizer to specify the order of join tables as part of its optimization process. The join occurs in the order in which the tables were specified in the query.
IGNORE_LIKE_ REDUNDANT_SHIFTS Specifies whether redundant shift characters are ignored for DBCS-Open operands when processing the SQL LIKE predicate or OPNQRYF command %WLDCRD built-in function.	*DEFAULT	The default value is set to *OPTIMIZE.
	*ALWAYS	When processing the SQL LIKE predicate or **OPNQRYF** command %WLDCRD built-in function, redundant shift characters are ignored for DBCS-Open operands. The optimizer cannot use an index to perform key row positioning for SQL LIKE or **OPNQRYF** %WLDCRD predicates involving DBCS-Open, DBCS-Either, or DBCS-Only operands.
	*OPTIMIZE	When processing the SQL LIKE predicate or the **OPNQRYF** command %WLDCRD built-in function, redundant shift characters might be ignored for DBCS-Open operands. These characters are ignored depending on whether an index is used to perform key row positioning for these predicates. This option enables the query optimizer to consider key row positioning for SQL LIKE or **OPNQRYF** %WLDCRD predicates involving DBCS-Open, DBCS-Either, or DBCS-Only operands.

Parameter	Value	Description
LIMIT_PREDICATE_ OPTIMIZATION Specifies that the query optimizer can only use simple isolatable predicates (OIF) when performing its index optimization. An OIF is a predicate that can eliminate a record without further evaluation. Any predicate that cannot be classified as an OIF is ignored by the optimizer and needs to be evaluated as a non-key selection predicate. A=10 and (A => 10 AND B=9) are OIFs. A=10 OR B=9 are not OIFs. Note: *YES impairs or limits index optimization.	*DEFAULT	Do not eliminate the predicates that are not simple isolatable predicates (OIF) when doing index optimization. Same as *NO.
	*NO	Do not eliminate the predicates that are not simple isolatable predicates (OIF) when doing index optimization.
	*YES	Eliminate the predicates that are not simple isolatable predicates (OIF) when doing index optimization.
LOB_LOCATOR_THRESHOLD Specifies either *DEFAULT or an Integer Value -- the threshold to free eligible LOB locators that exist within the job.	*DEFAULT	The default value is set to 0. This option indicates that the database does not free locators.
	Integer Value	If the value is 0, then the database does not free locators. For values 1 through 250,000, on a FETCH request, the database compares the SQL current LOB locator count for the job against the threshold value. If the locator count is greater than or equal to the threshold, the database frees host server created locators that have been retrieved. This option applies to all host server jobs (QZDASOINIT) and has no impact to other jobs.
MATERIALIZED_QUERY_ TABLE_REFRESH_AGE Specifies the usage of materialized query tables in query optimization and runtime.	*DEFAULT	The default value is set to 0.
	0	No materialized query tables can be used.
	*ANY	Any tables indicated by the MATERIALIZED_ QUERY_TABLE_USAGE INI parameter can be used.
	`Timestamp_ duration`	Only tables indicated by MATERIALIZED_ QUERY_TABLE_USAGE INI option which have a REFRESH TABLE performed within the specified timestamp duration can be used.
MATERIALIZED_QUERY_ TABLE_USAGE Specifies the ability to examine which materialized query tables are eligible to be used based on the last time a REFRESH TABLE statement was run.	*DEFAULT	The default value is set to *NONE.
	*NONE	Materialized query tables cannot be used in query optimization and implementation.
	*ALL	User-maintained materialized query tables may be used.
	*USER	User-maintained materialized query tables can be used.
MEMORY_POOL_PREFERENCE Specifies the preferred memory pool that database operations uses. This option does not guarantee use of the specified pool, but directs database to perform its paging into this pool when supported by the database operation.	*DEFAULT	The default value is set to *JOB.
	*JOB	Paging is done in the pool of the job. This option is normal paging behavior.
	*BASE	Attempt to page storage into the base pool when paging is needed and a database operation that supports targeted paging occurs.
	nn	Attempt to page storage into pool nn when paging is needed and a database operation that supports targeted paging occurs.
	*NAME PoolName	Attempt to page storage into a named storage pool when paging is needed and a database operation that supports targeted paging occurs.
	*PRIVATE Library/ Subsystem/ PoolNumber	Attempt to page storage into a private storage pool in specified library and subsystem when paging is needed and a database operation that supports targeted paging occurs.
MESSAGES_DEBUG Specifies whether Query Optimizer debug messages are displayed to the job log. These messages are regularly issued when the job is in debug mode.	*DEFAULT	The default is set to *NO.
	*NO	No debug messages are to be displayed.
	*YES	Issue all debug messages that are generated for **STRDBG**.
NORMALIZE_DATA Specifies whether normalization is performed on Unicode constants, host variables, parameter markers, and expressions that combine strings.	*DEFAULT	The default is set to *NO.
	*NO	Unicode constants, host variables, parameter markers, and expressions that combine strings is not normalized.
	*YES	Unicode constants, host variables, parameter markers, and expressions that combine strings is normalized

Table 46. Query Options Specified on QAQQINI Command (continued)

Table 46. Query Options Specified on QAQQINI Command (continued)		
Parameter	**Value**	**Description**
OPEN_CURSOR_CLOSE_ COUNT Specifies either *DEFAULT or an Integer Value: the number of cursors to full close when the threshold is encountered.	*DEFAULT	*DEFAULT is equivalent to 0. See Integer Value for details.
	Integer Value	This value determines the number of cursors to be closed. The valid values for this parameter are 1 - 65536. The value for this parameter is less than or equal to the number in the OPEN_CURSOR_THREHOLD parameter. If the number of open cursors reaches the value specified by the OPEN_CURSOR_THRESHOLD, pseudo-closed cursors are hard (fully) closed. The least recently used cursors are closed first. This value is ignored if OPEN_CURSOR_THRESHOLD is *DEFAULT. If OPEN_CURSOR_THRESHOLD is specified and the value is *DEFAULT, the number of cursors closed is equal to OPEN_CURSOR_THRESHOLD multiplied by 10 percent. The result is rounded up to the next integer value. OPEN_CURSOR_CLOSE_COUNT is used with OPEN_CURSOR_THRESHOLD to manage the number of open cursors within a job. Open cursors include pseudo-closed cursors.
OPEN_CURSOR_ THRESHOLD Specifies either *DEFAULT or an Integer Value -- the threshold to start full close of pseudo-closed cursors.	*DEFAULT	*DEFAULT is equivalent to 0. See Integer Value for details.
	Integer Value	This value determines the threshold to start full close of pseudo-closed cursors. When the number of open cursors reaches this threshold value, pseudo-closed cursors are hard (fully) closed with the least recently used cursors being closed first. The number of cursors to be closed is determined by OPEN_CURSOR_CLOSE_COUNT. The valid user-entered values for this parameter are 1 - 65536. A default value of 0 indicates that there is no threshold. Hard closes are not forced based on the number of open cursors within a job. OPEN_CURSOR_THRESHOLD is used with OPEN_CURSOR_CLOSE_COUNT to manage the number of open cursors within a job. Open cursors include pseudo-closed cursors.
OPTIMIZATION_GOAL Specifies the goal that the query optimizer uses when making costing decisions.	*DEFAULT	Optimization goal is determined by the interface (ODBC, SQL precompiler options, `OPTIMIZE FOR nnn ROWS` clause).
	*FIRSTIO	All queries are optimized with the goal of returning the first page of output as fast as possible. This option works well when the output is controlled by a user likely to cancel the query after viewing the first page of data. Queries coded with `OPTIMIZE FOR nnn ROWS` honor the goal specified by the clause.
	*ALLIO	All queries are optimized with the goal of running the entire query to completion in the shortest amount of elapsed time. This option is better when the output of a query is written to a file or report, or the interface is queuing the output data. Queries coded with `OPTIMIZE FOR nnn ROWS` honor the goal specified by the clause.
OPTIMIZE_STATISTIC_ LIMITATION Specifies limitations on the statistics gathering phase of the query optimizer. One of the most time consuming aspects of query optimization is in gathering statistics from indexes associated with the queried tables. Generally, the larger the size of the tables involved in the query, the longer the gathering phase of statistics takes. This option provides the ability to limit the amount of resources spend during this phase of optimization. The more resources spent on statistics gathering, the more accurate (optimal) the optimization plan is.	*DEFAULT	The amount of time spent in gathering index statistics is determined by the query optimizer.
	*NO	No index statistics are gathered by the query optimizer. Default statistics are used for optimization. (Use this option sparingly.)
	*PERCENTAGE integer value	Specifies the maximum percentage of the index that is searched while gathering statistics. Valid values for are 1 - 99.
	*MAX_ NUMBER_OF_ RECORDS_ ALLOWED integer value	Specifies the largest table size, in number of rows, for which gathering statistics is allowed. For tables with more rows than the specified value, the optimizer does not gather statistics and uses default values.

<table>
<caption>Table 46. Query Options Specified on QAQQINI Command (continued)</caption>
<tr><th>Parameter</th><th>Value</th><th>Description</th></tr>
<tr><td rowspan="6">PARALLEL_DEGREE

Specifies the parallel processing option that can be used when running database queries and database file keyed access path builds, rebuilds, and maintenance in the job. The specified parallel processing option determines the types of parallel processing allowed. There are two types of parallel processing:
1. Input/Output (I/O) parallel processing. With I/O parallel processing, the database manager uses multiple tasks for each query to do the I/O processing. The central processor unit (CPU) processing is still done serially.
2. Symmetric Multiprocessing (SMP). SMP assigns both CPU and I/O processing to tasks that run the query in parallel. Actual CPU parallelism requires a system with multiple processors. SMP can only be used if the system feature, DB2 Symmetric Multiprocessing, is installed. Use of SMP parallelism can affect the order in which records are returned.</td><td>*DEFAULT</td><td>The default value is *SYSVAL.</td></tr>
<tr><td>*SYSVAL</td><td>Set to the current system value QQRYDEGREE.</td></tr>
<tr><td>*IO</td><td>Any number of tasks can be used. SMP parallel processing is not allowed.</td></tr>
<tr><td>*OPTIMIZE</td><td>Any number of tasks for:
• I/O or SMP parallel processing of the query
• database file keyed access path build, rebuild, or maintenance.

SMP parallel processing is used only if the system feature, DB2 Symmetric Multiprocessing for IBM i, is installed.

Use of parallel processing and the number of tasks used is determined by:
• the number of processors available in the system
• the job share of the amount of active memory available in the pool in which the job is run
• whether the expected elapsed time for the query or database file keyed access path build or rebuild is limited by CPU processing or I/O resources.

The query optimizer chooses an implementation that minimizes elapsed time based on the job share of the memory in the pool.</td></tr>
<tr><td>*OPTIMIZE nnn</td><td>Like *OPTIMIZE, with the value nnn indicating a percentage from 1 to 200, used to influence the number of tasks. If not specified, 100 is used.

The query optimizer determines the parallel degree for the query using the same processing as is done for *OPTIMIZE. Once determined, the optimizer adjusts the actual parallel degree used for the query by the percentage given.

Allows the user to override the parallel degree used without having to specify a particular parallel degree under *NUMBER_OF_TASKS.</td></tr>
<tr><td>nnn</td><td>The query optimizer chooses to use either I/O or SMP parallel processing to process the query. SMP parallel processing is used only if the system feature, DB2 Symmetric Multiprocessing for IBM i, is installed.

nnn is a percentage from 1 to 200 and is used to influence the number of tasks. If not specified, 100 is used.

The choices made by the query optimizer are like those choices made for parameter value *OPTIMIZE. The exception is the assumption that all pool active memory can be used for query processing, database file keyed access path build, rebuild, or maintenance.</td></tr>
<tr><td rowspan="3">PARALLEL_DEGREE (continued)</td><td>*NONE</td><td>No parallel processing is allowed for database query processing or database table index build, rebuild, or maintenance.</td></tr>
<tr><td>*NUMBER_ OF _TASKS nnn</td><td>Indicates the maximum number of tasks that can be used for a single query. The number of tasks is limited to either this value or the number of disk arms associated with the table.

Not recommended if running SQE. The SQE optimizer attempts to use this degree and override many of the normal costing mechanisms. For SQE, use *OPTIMIZE with a percentage.</td></tr>
<tr><td>*MAX xxx</td><td>Like *MAX, with the value xxx indicating the ability to specify an integer percentage value 1 - 200. The query optimizer determines the parallel degree for the query using the same processing as is done for *MAX. Once determined, the optimizer adjusts the actual parallel degree used for the query by the percentage given. This option provides the user the ability to override the parallel degree used to some extent without having to specify a particular parallel degree under *NUMBER_OF_TASKS.</td></tr>
<tr><td rowspan="3">PARAMETER_MARKER_ CONVERSION

Specifies whether to allow literals to be implemented as parameter markers in dynamic SQL queries.</td><td>*DEFAULT</td><td>The default value is set to *YES.</td></tr>
<tr><td>*NO</td><td>Constants cannot be implemented as parameter markers.</td></tr>
<tr><td>*YES</td><td>Constants can be implemented as parameter markers.</td></tr>
</table>

Parameter	Value	Description
PREVENT_ADDITIONAL_CONFLICTING_LOCKS The following SQL DDL statements require an exclusive, no read lock on the target table. If the application activity cannot be quiesced, it can be hard to accomplish these operations. The PREVENT_ADDITIONAL_CONFLICTING_LOCKS QAQQINI option provides a control for customers to use to direct the operating system to favor a request for an exclusive, no read lock over new requests to lock the object for reading.	*DEFAULT	The default value is set to *NO
	*NO	When a job requests an exclusive lock on an object, do not prevent concurrent jobs from acquiring additional locks on the object.
	*YES	When *YES is chosen, any new requests for these lower-level read locks will be kept behind the exclusive lock request and could surface to applications as the table is unavailable for use for querying. • ALTER TABLE (Add, Alter or Drop Column) • CREATE TRIGGER • LOCK TABLE • RENAME TABLE
PSEUDO_OPEN_CHECK_HOST_VARS This option can be used to allow SQE to check the selectivity of the host variable values at pseudo open time. If the new set of host variable values require a different plan to perform well, SQE will re-optimize the query. This option is most appropriate when there is considerable variability in the selectivity of host variable in the queries predicates.	*DEFAULT	The default value is set to *NO
	*NO	The optimizer does not check host variables for selectivity changes once in pseudo-open.
	*OPTIMIZE	The optimizer will determine when a host variable selectivity should be checked. In general, the optimizer will monitor the query and if after a certain number of runs it determines that there is no advantage to checking host variable selectivity at pseudo open time, it will stop checking. Full opens do normal plan validation.
	*YES	The optimizer will always check host variable selectivity at pseudo open time. **Note:** If the REOPTIMIZE_ACCESS_PLAN INI option is set to *ONLY_REQUIRED then this INI option has no effect.
QUERY_TIME_LIMIT Specifies a time limit for database queries allowed to be started based on the estimated number of elapsed seconds that the query requires to process.	*DEFAULT	The default value is set to *SYSVAL.
	*SYSVAL	The query time limit for this job is obtained from the system value, QQRYTIMLMT.
	*NOMAX	There is no maximum number of estimated elapsed seconds.
	integer value	Specifies the maximum value that is checked against the estimated number of elapsed seconds required to run a query. If the estimated elapsed seconds are greater than this value, the query is not started. Valid values range from 0 to 2,147,352,578.
REOPTIMIZE_ACCESS_PLAN Specifies whether the query optimizer reoptimizes a query with a saved access plan. Queries can have a saved access plan stored in the associated storage of an HLL program, or in the plan cache managed by the optimizer itself. Note: If you specify *NO the query could still be revalidated. Some of the reasons this option might be necessary are: • The queried file was deleted and recreated. • The query was restored to a different system than the one on which it was created. • An OVRDBF command was used.	*DEFAULT	The default value is set to *NO.
	*NO	Do not force the existing query to be reoptimized. However, if the optimizer determines that optimization is necessary, the query is optimized.
	*YES	Force the existing query to be reoptimized.
	*FORCE	Force the existing query to be reoptimized.
	*ONLY_ REQUIRED	Do not allow the plan to be reoptimized for any subjective reasons. For these cases, continue to use the existing plan since it is still a valid workable plan. This option could mean that you might not get all the performance benefits that a reoptimization plan might derive. Subjective reasons include file size changes, new indexes, and so on. Non-subjective reasons include deletion of an index used by existing access plan, query file being deleted and recreated, and so on.
SQE_NATIVE_ACCESS This option controls how native access will be implemented for an open or query. It does not affect a simple native open, such as an open done using the OPNDBF command, unless opening an SQL view, a partition table with a MBR(*ALL) override or a file dependent on row or column access control. It also does not affect the Query (QQQQRY) API.	*DEFAULT	The default value could be either *YES or *NO as determined by the Query Optimizer.
	*NO	Attempt open using the Classic Query Engine (CQE). If CQE is not possible, attempt open using the SQL Query Engine (SQE).
	*YES	Attempt open using the SQL Query Engine (SQE). If SQE is not possible, attempt open using the Classic Query Engine (CQE).

Table 46. Query Options Specified on QAQQINI Command (continued)

Table 46. Query Options Specified on QAQQINI Command (continued)		
Parameter	**Value**	**Description**
SQE_NATIVE_ACCESS_POSITION_BEHAVIOR This option controls the positioning behavior of native opens or queries implemented by SQE. By specifying an option other than *DEFAULT, performance benefits may be realized.	*DEFAULT	Normal positioning behavior is performed.
	*NO_ROLLBACK_HOLD	The current cursor position is unchanged by a rollback.
	*NO_KEY_FAILURE_HOLD	If an attempted key positioning operation fails,the cursor position prior to the attempted operation will not be restored. It is assumed that another absolute positioning operation, such as first, last, or key equal, will be attempted before any relative positioning operations, such as next or previous.
	*NO_HOLD	Behavior is the same as defined for *NO_ROLLBACK_HOLD and *NO_KEY_FAILURE_HOLD values.
SQLSTANDARDS_MIXED_ CONSTANT Specifies whether to allow IGC constants to always be treated as IGC-OPEN in SQL queries. **Note:** When *NO is specified, DB2 for i is not compatible with the other DB2 platforms.	*DEFAULT	The default value is set to *YES.
	*YES	SQL IGC constants are treated as IGC-OPEN constants.
	*NO	If the data in the IGC constant only contains shift-out DBCS-data shift-in, then the constant are treated as IGC-ONLY, otherwise it is treated as IGC-OPEN.
SQL_CONCURRENT_ACCESS_RESOLUTION Specifies the concurrent access resolution to use for an SQL query.	*DEFAULT	The default value is set to *WAIT.
	*WAIT	The database manager must wait for the commit or rollback when encountering data in the process of being updated, deleted, or inserted. Rows encountered that are in the process of being inserted are not skipped. This option applies if possible when the isolation level in effect is Cursor Stability or Read Stability and is ignored otherwise.
	*CURCMT	The database manager can use the currently committed version of the data for read-only scans when it is in the process of being updated or deleted. Rows in the process of being inserted can be skipped. This option applies if possible when the isolation level in effect is Cursor Stability and is ignored otherwise.
SQL_DECFLOAT_WARNINGS Specifies the warnings returned for SQL DECFLOAT computations and conversions involving: • division by 0. • overflow. • underflow. • an invalid operand. • an inexact result. •	*DEFAULT	The default value is set to *NO.
	*YES	A warning is returned to the caller for DECFLOAT computations and conversions involving division by 0, overflow, underflow, invalid operand, inexact result, or subnormal number.
	*NO	An error or a mapping error is returned to the caller for DECFLOAT computations and conversions involving division by 0, overflow, underflow, or an invalid operand. A warning or error is not returned for an inexact result or a subnormal number.
SQL_FAST_DELETE_ ROW_COUNT Specifies how the delete is implemented by the database manager. This value is used when processing a DELETE FROM table-name SQL statement without a WHERE clause.	*DEFAULT	The default value is set to 0. 0 indicates that the database manager chooses how many rows to consider when determining whether fast delete could be used instead of traditional delete. When using the default value, the database manager will most likely use 1000 as a row count. This means that using the INI option with a value of 1000 results in no operational difference from using 0 for the option.
	*NONE	This value forces the database manager to never attempt to fast delete on the rows.
	*OPTIMIZE	This value is same as using *DEFAULT.
	Integer Value	Specifying a value for this option allows the user to tune the behavior of DELETE. The target table for the DELETE statement must match or exceed the number of rows specified on the option for fast delete to be attempted. A fast delete does not write individual rows into a journal. The valid values are 1 - 999,999,999,999,999.

Table 46. Query Options Specified on QAQQINI Command (continued)		
Parameter	**Value**	**Description**
SQL_GVAR_BUILD_RULE Determines whether global variables must exist or not when building SQL routines or executing SQL pre-compiles. This option has no affect on dynamic SQL statements.	*DEFAUT	The default value is set to *DEFER
	*DEFER	Global variables do not need to exist when an SQL routine is created or the SQL pre-compiler is run. Since global variables are not required to exist, the create will not fail when an incorrect column name or routine variable is encountered. Incorrect name usage will result in SQL0206 - "Column or global variable &1 not found." failures when the statement is executed.
	*EXIST	Global variables referenced by SQL must exist when the SQL routine is created or the SQL pre-compiler is run. Using this option, an SQL0206 will be issued at create time.
SQL_MODIFIES_SQL_DATA From the SQL Standard, no MODIFIES SQL DATA operations are allowed in an SQL BEFORE trigger. The Informix® database allows MODIFIES SQL DATA operations in SQL BEFORE triggers. Setting the option to *YES allows SQL BEFORE triggers to perform the SQL MODIFIES SQL DATA operations.	*DEFAULT	The default value is set to *NO.
	*NO	No MODIFIES SQL DATA operations are allowed in an SQL BEFORE trigger.
	*YES	MODIFIES SQL DATA operations are allowed in an SQL BEFORE trigger.
SQL_PSEUDO_CLOSE Before V6R1: SQL cursor open processing checks for the presence of a data area named QSQPSCLS1 in the library list of the job. If the data area is found, all reusable cursors are marked as candidates for reuse. They are pseudo-closed the first time rather than the second time the application closes the cursor. Without this data area, a cursor does not become reusable until the second close. Pseudo-closing the first time results in leaving some cursors open that might not be reused. These open cursors can increase the amount of auxiliary and main storage required for the application. The storage can be monitored using the WRKSYSSTS command. For the amount of auxiliary storage used, look at the "% system ASP used." For the amount of main storage, examine the faulting rates on the WRKSYSSTS display. The format and the contents of the data area are not important. The data area can be deleted using the following command: DLTDTAARA DTAARA(QGPL/QSQPSCLS1). The existence of the data area is checked during the first SQL open operation for each job. It is checked only once and the processing mode remains the same for the life of the job. Because the library list is used for the search, the change can be isolated to specific jobs. Create the data area in a library that is included only in the library lists for those jobs.	*DEFAULT	The default behavior depends upon whether the QSQPSCLS1 *DTAARA exists. If the QSQPSCLS1 *DTAARA was found on the first OPEN within the job, then SQL cursors are marked as candidates for reuse. The cursors are pseudo-closed on the first close. If the QSQPSCLS1 *DTAARA was not found on the first OPEN within the job, then SQL cursors are marked as candidates for reuse. The cursors are pseudo-closed on the second close.
	Integer Value	Specifies a value greater than zero that indicates when a cursor is pseudo-closed. The value of this option minus 1 indicates how many times the cursor is hard closed before being marked as candidate for pseudo-close. Valid values are 1 - 65535.
SQL_STMT_COMPRESS_MAX Specifies the compression maximum setting, which is used when statements are prepared into a package.	*DEFAULT	The default value is set to 2. The default indicates that the access plan associated with any statement will be removed after a statement has been compressed twice without being executed.
	Integer Value	The integer value represents the number of times that a statement is compressed before the access plan is removed to create more space in the package. Executing the SQL statement resets the count for that statement to 0. The valid Integer values are 1 - 255.
SQL_STMT_REUSE Specifies the number of times the statement must be prepared in the same connection before the statement is stored in the SQL extended dynamic package. If the number of times the statement has been prepared in the same connection is less than the specified INI option, a temporary copy of the statement is used. Any other job preparing the statement does a complete prepare.	*DEFAULT	The default value is 3. The statement is stored on the third prepare of the statement.
	0	The statement will be stored on the first prepare of the statement. This was the default behavior prior to V6R1M0
	1::255	The number of times the statement must be prepared in the same connection before the statement is stored in the SQL package.
SQL_SUPPRESS_MASKED_DATA_DETECTION	*DEFAULT	The default value is set to *NO.
	*YES	If masked data is being used to insert into or update a table, detection of this masked data will not be done and a SQ20478 with reason code 30 will not be sent.
	*NO	If masked data is being used to insert into or update a table with activated column access control directly from an expression involving a column with an active column mask, detection of this masked data will be done and a SQ20478 with reason code 30 will be sent.

Table 46. Query Options Specified on QAQQINI Command (continued)		
Parameter	**Value**	**Description**
SQL_SUPPRESS_WARNINGS For SQL statements, this parameter provides the ability to suppress SQL warnings.	*DEFAULT	The default value is set to *NO.
	*YES	Examine the SQLCODE in the SQLCA after execution of a statement. If the SQLCODE is + 30, then alter the SQLCA so that no warning is returned to the caller. Set the SQLCODE to 0, the SQLSTATE to '00000' and SQLWARN to ' '. Warnings: • SQL0335 • SQL0030 • SQL7909 (on a DROP PROCEDURE/ROUTINE/FUNCTION)
	*NO	Specifies that SQL warnings are returned to the caller.
SQL_TRANSLATE_ASCII_ TO_JOB Specifies whether to translate SQL statement text on the application server (AS) according to the CCSID of the job. This option applies when using DRDA to connect to an IBM i as the AS where the application requestor (AR) machine is an ASCII-based platform.	*DEFAULT	The default value is set to *NO.
	*YES	Translate ASCII SQL statement text to the CCSID of the IBM i job.
	*NO	Translate ASCII SQL statement text to the EBCIDIC CCSID associated with the ASCII CCSID.
SQL_XML_DATA_CCSID Specifies the CCSID to be used for XML columns, host variables, parameter markers, and expressions, if not explicitly specified. See "SQL_XML_DATA_CCSID QAQQINI option" on page 187	*DEFAULT	The default value is set to 1208.
	*JOB	The job CCSID is used for XML columns, host variables, parameter markers, and expressions, if not explicitly specified. If the job CCSID is 65535, the default CCSID of 1208 is used.
	Integer Value	The CCSID used for XML columns, host variables, parameter markers, and expressions, if not explicitly specified. This value must be a valid single-byte or mixed EBCDIC CCSID or Unicode CCSID. The value cannot be 65535.
STAR_JOIN **Note:** Only modifies the environment for the Classic Query Engine. Specifies enhanced optimization for hash queries where both a hash join table and a Distinct List of values is constructed from the data. This Distinct List of values is appended to the selection against the primary table of the hash join Any EVI indexes built over these foreign key columns can be used to perform bitmap selection against the table before matching the join values. The use of this option does not guarantee that star join is chosen by the optimizer. It only allows the use of this technique if the optimizer has decided to implement the query by using a hash join.	*DEFAULT	The default value is set to *NO
	*NO	The EVI Star Join optimization support is not enabled.
	*COST	Allow query optimization to cost the usage of EVI Star Join support. The optimizer determines whether the Distinct List selection is used based on how much benefit can be derived from using that selection.
STORAGE_LIMIT Specifies a temporary storage limit for database queries. If a query is expected to use more than the specified amount of storage, the query is not allowed to run. The value specified is in megabytes.	*DEFAULT	The default value is set to *NOMAX.
	*NOMAX	Never stop a query from running because of storage concerns.
	Integer Value	The maximum amount of temporary storage in megabytes that can be used by a query. This value is checked against the estimated amount of temporary storage required to run the query as calculated by the query optimizer. If the estimated amount of temporary storage is greater than this value, the query is not started. Valid values range from 0 through 2147352578.
SYSTEM_SQL_STATEMENT_ CACHE Specifies whether to disable the system-wide SQL Statement Cache for SQL queries.	*DEFAULT	The default value is set to *YES.
	*YES	Examine the system-wide SQL Statement Cache when an SQL prepare request is processed. If a matching statement exists in the cache, use the results of that prepare. This option allows the application to potentially have better performing prepares.
	*NO	Specifies that the system-wide SQL Statement Cache is not examined when processing an SQL prepare request.

Table 46. Query Options Specified on QAQQINI Command (continued)		
Parameter	**Value**	**Description**
TEXT_SEARCH_DEFAULT_TIMEZONE Specifies the time zone to apply to any date or dateTime value specified in an XML text search using the CONTAINS or SCORE function. The time zone is the offset from UTC (Greenwich mean time. It is only applicable when a specific time zone is not given for the value.	*DEFAULT	Use the default as defined by database. This option is equivalent to UTC.
	sHH:MM	A time zone formatted value where • s is the sign, + or – • HH is the hour • MM is the minute The valid range for HH is 00 - 23. The valid range for MM is 00 - 59. The format is specific. All values are required, including sign. If HH or MM is less than 10, it must have a leading zero specified.
UDF_TIME_OUT **Note:** Only modifies the environment for the Classic Query Engine. Specifies the amount of time, in seconds, that the database waits for a User Defined Function (UDF) to finish processing.	*DEFAULT	The amount of time to wait is determined by the database. The default is 30 seconds.
	*MAX	The maximum amount of time that the database waits for the UDF to finish.
	integer value	Specify the number of seconds that the database waits for a UDF to finish. If the value given exceeds the database maximum wait time, the maximum wait time is used by the database. Minimum value is 1 and maximum value is system defined.
VARIABLE_LENGTH_ OPTIMIZATION Specifies whether aggressive optimization techniques are used on variable length columns.	*DEFAULT	The default value is set to *YES.
	*YES	Enables aggressive optimization of variable-length columns, including index-only access. It also allows constant value substitution when an equal predicate is present against the columns. As a consequence, the length of the data returned for the variable-length column might not include any trailing blanks that existed in the original data. As a result, the application can receive the substituted value back instead of the original data. Function calls could operate on the substituted value instead of the original string value.
	*NO	Do not allow aggressive optimization of variable length columns.

Note: The following QAQQINI options will be ignored for SQE native query access. These options were previously honored for CQE native query access.

- LIMIT_PREDICATE_OPTIMIZATION
- STAR_JOIN
- UDF_TIME_OUT

Note: The following QAQQINI options will be honored for SQE native query access. These options were previously ignored for CQE native query access.

- DETERMINISTIC_UDF_SCOPE
- FIELDPROC_ENCODED_COMPARISON
- MATERIALIZED_QUERY_TABLE_USAGE
- MATERIALIZED_QUERY_TABLE_REFRESH_AGE
- MEMORY_POOL_PREFERENCE
- VARIABLE_LENGTH_OPTIMIZATION

SQL_XML_DATA_CCSID QAQQINI option

The SQL_XML_DATA_CCSID QAQQINI option has several settings that affect SQL processing.

The SQL_XML_DATA_CCSID QAQQINI setting is applied within SQL in the following SQL processing:

Table 47. SQL_XML_DATA_CCSID setting application within SQL	
SQL Processing item	**Description**
Valid values for the QAQQINI option are CCSIDs allowed on an XML column.	Valid values are all EBCDIC SBCS and mixed CCSIDs, and Unicode 1208, 1200, and 13488 CCSIDs.

Table 47. SQL_XML_DATA_CCSID setting application within SQL (continued)	
SQL Processing item	**Description**
Does not affect the promotion of SQL data types.	Other SQL data types cannot be directly promoted to the SQL XML data type.
XMLPARSE untyped parameter markers.	The QAQQINI setting applies to untyped parameter markers passed as string-expression. The type is CLOB(2G) for SBCS, mixed, and UTF-8 values. The type is DBCLOB(1G) for Unicode 1200 and 13488.
XMLCOMMENT, XMLTEXT, XMLPI untyped parameter markers.	The QAQQINI setting applies to untyped parameter markers passed as string-expression. The type is VARCHAR(32740) for SBCS, mixed, and UTF-8 values. The type is VARGRAPHIC(16370) for Unicode 1200 and 13488.
Applies to parameter marker casts to the XML type for XMLCONCAT, and XMLDOCUMENT.	Applies to an untyped parameter marker passed as an XML-expression. Unless an explicit CCSID clause is specified, the CCSID of the parameter marker is obtained from the QAQQINI setting.
The QAQQINI setting does not affect storage and retrieval assignment rules.	The CCSID of the host variables and table columns apply.
String to column assignment on SQL INSERT and UPDATE.	An implicit or explicit XMLPARSE is required on the column assignment.
String to host variable assignment.	An implicit or explicit XMLSERIALIZE is required on the host variable assignment.
Column to column assignment.	When the target column is XML, an implicit XMLPARSE is applied if the source column is not XML. The target XML column has a defined XML CCSID. When the source column is XML, an explicit XMLSERIALIZE is required if the target column is not XML.
Host variable to column assignment.	The target column has a defined CCSID.
UNION ALL (if XML publishing functions in query).	The XML result CCSID is obtained from the QAQQINI setting.
Does not apply to SQL constants.	UX constants are defined as UTF-16. FX constants are defined as UTF-8.
Result type of XML data built-in functions.	If the first operand of XMLPARSE and XMLVALIDATE is an untyped parameter marker, the CCSID is set from the QAQQINI setting, which then affects the XML result CCSID. The QAQQINI setting is used for XMLSERIALIZE for CHAR, VARCHAR, and LOB AS data-type. UTF-16 is used for GRAPHIC, DBCLOB, and NCHAR.
Result type of XML publishing functions - XMLAGG, XMLGROUP, XMLATTRIBUTES, XMLCOMMENT, XMLCONCAT, XMLDOCUMENT, XMELEMENT, XMLFOREST, XMLNAMESPACES, XMLPI, XMLROW, and XMLTEXT.	The XML result CCSID for XML publishing functions is obtained from the QAQQINI setting.
Result type of XML publishing functions in a view.	The XML result CCSID is set when the view is created.
XML data type on external procedure XML AS parameters.	The XML parameter CCSID is set when the procedure is created.

Table 47. SQL_XML_DATA_CCSID setting application within SQL (continued)	
SQL Processing item	**Description**
XML data type on external user-defined functions.	The XML parameter and result CCSID are set when the function is created.
CREATE TABLE XML column.	The QAQQINI setting is used for dynamic SQL. The QAQQINI setting is set in *PGM, *SRVPGM, and *SQLPKG objects when created.
MQTs containing select-statement with XML publishing functions.	The CCSID is set when the MQT is created. The CCSID is maintained for an ALTER TABLE.
ALTER TABLE ADD MATERIALIZED QUERY definition.	The QAQQINI setting is used if the select-statement contains XML publishing functions.
XML AS CLOB CCSID	The QAQQINI setting is built into *PGM and *SRVPGM objects when the program is created. The CCSID defaults to UTF-8 for CLOB when QAQQINI setting is UTF-16 or UCS2.
XML AS DBCLOB CCSID	The default for DBCLOB is always UTF-16 for XML.
SQL GET and SET DESCRIPTOR XML data type.	QAQQINI setting applied to XML data type.
SQL Global variables.	QAQQINI setting applied to global variables with the XML data type.

Related information

XML values

SQL statements and SQL/XML functions

Setting resource limits with the Predictive Query Governor

The Db2 for i Predictive Query Governor can stop the initiation of a query if the estimated run time (elapsed execution time) or estimated temporary storage for the query is excessive. The governor acts *before* a query is run instead of while a query is run. The governor can be used in any interactive or batch job on the system. It can be used with all Db2 for i query interfaces and is not limited to use with SQL queries.

The ability of the governor to predict and stop queries before they are started is important because:

- Operating a long-running query and abnormally ending the query before obtaining any results wastes system resources.
- Some CQE operations within a query cannot be interrupted by the **End Request (ENDRQS)** CL command. The creation of a temporary index or a query using a column function without a GROUP BY clause are two examples of these types of queries. It is important to not start these operations if they take longer than the user wants to wait.

The governor in Db2 for i is based on two measurements:

- The estimated runtime for a query.
- The estimated temporary storage consumption for a query.

If the query estimated runtime or temporary storage usage exceed the user-defined limits, the initiation of the query can be stopped.

To define a time limit (in seconds) for the governor to use, do one of the following:

- Use the Query Time Limit (QRYTIMLMT) parameter on the **Change Query Attributes (CHGQRYA)** CL command. The command language used is the first place where the optimizer attempts to find the time limit.

- Set the Query Time Limit option in the query options file. The query options file is the second place where the query optimizer attempts to find the time limit.
- Set the QQRYTIMLMT system value. Allow each job to use the value *SYSVAL on the **Change Query Attributes (CHGQRYA)** CL command, and set the query options file to *DEFAULT. The system value is the third place where the query optimizer attempts to find the time limit.

To define a temporary storage limit (in megabytes) for the governor to use, do the following:

- Use the Query Storage Limit (QRYSTGLMT) parameter on the **Change Query Attributes (CHGQRYA)** CL command. The command language used is the first place where the query optimizer attempts to find the limit.
- Set the Query Storage Limit option STORAGE_LIMIT in the query options file. The query options file is the second place where the query optimizer attempts to find the time limit.

The time and temporary storage values generated by the optimizer are *only* estimates. The actual query runtime might be more or less than the estimate. In certain cases when the optimizer does not have full information about the data being queried, the estimate could vary considerably from the actual resource used. In those cases, you might need to artificially adjust your limits to correspond to an inaccurate estimate.

When setting the time limit for the entire system, set it to the maximum allowable time that any query must be allowed to run. By setting the limit too low you run the risk of preventing some queries from completing and thus preventing the application from successfully finishing. There are many functions that use the query component to internally perform query requests. These requests are also compared to the user-defined time limit.

You can check the inquiry message CPA4259 for the predicted runtime and storage. If the query is canceled, debug messages are still written to the job log.

You can also add the Query Governor Exit Program that is called when estimated runtime and temporary storage limits have exceeded the specified limits.

Related information

Query Governor Exit Program
End Request (ENDRQS) command
Change Query Attributes (CHGQRYA) command

Using the Query Governor

The resource governor works with the query optimizer.

When a user issues a request to the system to run a query, the following occurs:

1. The query access plan is created by the optimizer.

   As part of the evaluation, the optimizer predicts or estimates the runtime for the query. This estimate helps determine the best way to access and retrieve the data for the query. In addition, as part of the estimating process, the optimizer also computes the estimated temporary storage usage for the query.
2. The estimated runtime and estimated temporary storage are compared against the user-defined query limit currently in effect for the job or user session.
3. If the estimates for the query are less than or equal to the specified limits, the query governor lets the query run without interruption. No message is sent to the user.
4. If the query limit is exceeded, inquiry message CPA4259 is sent to the user. The message states the estimates as well as the specified limits. Realize that only one limit needs to be exceeded; it is possible that you see that only one limit was exceeded. Also, if no limit was explicitly specified by the user, a large integer value is shown for that limit.

   Note: A default reply can be established for this message so that the user does not have the option to reply. The query request is *always* ended.
5. If a default message reply is not used, the user chooses to do one of the following:
   - End the query request before it is run.

- Continue and run the query even though the estimated value exceeds the associated governor limit.

Setting the resource limits for jobs other than the current job

You can set either or both resource limits for a job other than the current job. You set these limits by using the JOB parameter on the **Change Query Attributes (CHGQRYA)** command. Specify either a query options file library to search (QRYOPTLIB) or a specific QRYTIMLMT, or QRYSTGLMT, or both for that job.

Using the resource limits to balance system resources

After the source job runs the **Change Query Attributes (CHGQRYA)** command, effects of the governor on the target job are not dependent upon the source job. The query resource limits remain in effect for the duration of the job or user session, or until a resource limit is changed by a **Change Query Attributes (CHGQRYA)** command.

Under program control, a user might be given different limits depending on the application function performed, time of day, or system resources available. These limits provide a significant amount of flexibility when trying to balance system resources with temporary query requirements.

Cancel a query with the Query Governor

When a query is expected to take more resources than the set limit, the governor issues inquiry message CPA4259.

You can respond to the message in one of the following ways:

- Enter a C to cancel the query. Escape message CPF427F is issued to the SQL runtime code. SQL returns SQLCODE -666.
- Enter an I to ignore the exceeded limit and let the query run to completion.

Control the default reply to the query governor inquiry message

The system administrator can control whether the interactive user has the option of ignoring the database query inquiry message by using the **Change Job (CHGJOB)** CL command.

Changes made include the following:

- If a value of *DFT is specified for the INQMSGRPY parameter of the **Change Job (CHGJOB)** CL command, the interactive user does not see the inquiry messages. The query is canceled immediately.
- If a value of *RQD is specified for the INQMSGRPY parameter of the **Change Job (CHGJOB)** CL command, the interactive user sees the inquiry. The user must reply to the inquiry.
- If a value of *SYSRPYL is specified for the INQMSGRPY parameter of the **Change Job (CHGJOB)** CL command, a system reply list is used to determine whether the interactive user sees the inquiry and whether a reply is necessary. The system reply list entries can be used to customize different default replies based on user profile name, user id, or process names. The fully qualified job name is available in the message data for inquiry message CPA4259. This algorithm allows the keyword CMPDTA to be used to select the system reply list entry that applies to the process or user profile. The user profile name is 10 characters long and starts at position 51. The process name is 10 character long and starts at position 27.
- The following example adds a reply list element that causes the default reply of C to cancel requests for jobs whose user profile is 'QPGMR'.

  ```
  ADDRPYLE  SEQNBR(56) MSGID(CPA4259) CMPDTA(QPGMR      51) RPY(C)
  ```

  The following example adds a reply list element that causes the default reply of C to cancel requests for jobs whose process name is 'QPADEV0011'.

  ```
  ADDRPYLE  SEQNBR(57) MSGID(CPA4259) CMPDTA(QPADEV0011 27) RPY(C)
  ```

Related information

Change Job (CHGJOB) command

Testing performance with the query governor

You can use the query governor to test the performance of your queries.

To test the performance of a query with the query governor, do the following:

1. Set the query time limit to zero (QRYTIMLMT(0)) using the **Change Query Attributes (CHGQRYA)** command or in the INI file. This forces an inquiry message from the governor stating that the estimated time to run the query exceeds the query time limit.
2. Prompt for message help on the inquiry message and find the same information that you can find by running the **Print SQL Information (PRTSQLINF)** command.

The query governor lets you optimize performance without having to run through several iterations of the query.

Additionally, if the query is canceled, the query optimizer evaluates the access plan and sends the optimizer debug messages to the job log. This process occurs even if the job is *not* in debug mode. You can then review the optimizer tuning messages in the job log to see if additional tuning is needed to obtain optimal query performance.

This method allows you to try several permutations of the query with different attributes, indexes, and syntax, or both. You can then determine what performs better through the optimizer without actually running the query to completion. This process saves on system resources because the actual query of the data is never done. If the tables to be queried contain many rows, this method represents a significant savings in system resources.

Be careful when you use this technique for performance testing, because all query requests are stopped before they are run. This caution is especially important for a CQE query that cannot be implemented in a single query step. For these types of queries, separate multiple query requests are issued, and then their results are accumulated before returning the final results. Stopping the query in one of these intermediate steps gives you only the performance information for that intermediate step, and not for the entire query.

Related information

Print SQL Information (PRTSQLINF) command

Change Query Attributes (CHGQRYA) command

Examples of setting query time limits

You can set the query time limit for the current job or user session using query options file QAQQINI. Specify the QRYOPTLIB parameter on the **Change Query Attributes (CHGQRYA)** command. Use a user library where the QAQQINI file exists with the parameter set to QUERY_TIME_LIMIT, and the value set to a valid query time limit.

To set the query time limit for 45 seconds you can use the following **Change Query Attributes (CHGQRYA)** command:

```
CHGQRYA   JOB(*) QRYTIMLMT(45)
```

This command sets the query time limit at 45 seconds. If the user runs a query with an estimated runtime equal to or less than 45 seconds, the query runs without interruption. The time limit remains in effect for the duration of the job or user session, or until the time limit is changed by the **Change Query Attributes (CHGQRYA)** command.

Assume that the query optimizer estimated the runtime for a query as 135 seconds. A message is sent to the user that stated that the estimated runtime of 135 seconds exceeds the query time limit of 45 seconds.

To set or change the query time limit for a job other than your current job, the **Change Query Attributes (CHGQRYA)** command is run using the JOB parameter. To set the query time limit to 45 seconds for job 123456/USERNAME/JOBNAME use the following **Change Query Attributes (CHGQRYA)** command:

```
CHGQRYA   JOB(123456/USERNAME/JOBNAME) QRYTIMLMT(45)
```

This command sets the query time limit at 45 seconds for job 123456/USERNAME/JOBNAME. If job 123456/USERNAME/JOBNAME tries to run a query with an estimated runtime equal to or less than 45 seconds the query runs without interruption. If the estimated runtime for the query is greater than 45 seconds, for example, 50 seconds, a message is sent to the user. The message states that the estimated runtime of 50 seconds exceeds the query time limit of 45 seconds. The time limit remains in effect for the duration of job 123456/USERNAME/JOBNAME, or until the time limit for job 123456/USERNAME/JOBNAME is changed by the **Change Query Attributes (CHGQRYA)** command.

To set or change the query time limit to the QQRYTIMLMT system value, use the following **Change Query Attributes (CHGQRYA)** command:

```
CHGQRYA QRYTIMLMT(*SYSVAL)
```

The QQRYTIMLMT system value is used for duration of the job or user session, or until the time limit is changed by the **Change Query Attributes (CHGQRYA)** command. This use is the default behavior for the **Change Query Attributes (CHGQRYA)** command.

Note: The query time limit can also be set in the INI file, or by using the **Change System Value (CHGSYSVAL)** command.

Related information

Change Query Attributes (CHGQRYA) command

Change System Value (CHGSYSVAL) command

Test temporary storage usage with the query governor

The predictive storage governor specifies a temporary storage limit for database queries. You can use the query governor to test if a query uses any temporary object, such as a hash table, sort, or temporary index.

To test for usage of a temporary object, do the following:

- Set the query storage limit to zero (QRYSTGLMT(0)) using the **Change Query Attributes (CHGQRYA)** command or in the INI file. This forces an inquiry message from the governor anytime a temporary object is used for the query. The message is sent regardless of the estimated size of the temporary object.
- Prompt for message help on the inquiry message and find the same information that you can find by running the **Print SQL Information (PRTSQLINF)** command. This command allows you to see what temporary objects were involved.

Related information

Print SQL Information (PRTSQLINF) command

Change Query Attributes (CHGQRYA) command

Examples of setting query temporary storage limits

The temporary storage limit can be specified either in the QAQQINI file or on the **Change Query Attributes (CHGQRYA)** command.

You can set the query temporary storage limit for a job using query options file QAQQINI. Specify the QRYOPTLIB parameter on the **Change Query Attributes (CHGQRYA)** command. Use a user library where the QAQQINI file exists with a valid value set for parameter STORAGE_LIMIT.

To set the query temporary storage limit on the **Change Query Attributes (CHGQRYA)** command itself, specify a valid value for the QRYSTGLMT parameter.

If a value is specified both on the **Change Query Attributes (CHGQRYA)** command QRYSTGLMT parameter and in the QAQQINI file specified on the QRYOPTLIB parameter, the QRYSTGLMT value is used.

To set the temporary storage limit for 100 MB in the current job, you can use the following **Change Query Attributes (CHGQRYA)** command:

```
CHGQRYA JOB(*) QRYSTGLMT(100)
```

If the user runs any query with an estimated temporary storage consumption equal to or less than 100 MB, the query runs without interruption. If the estimate is more than 100 MB, the CPA4259 inquiry message is sent by the database. To set or change the query time limit for a job other than your current job, the CHGQRYA command is run using the JOB parameter. To set the same limit for job 123456/USERNAME/JOBNAME use the following CHGQRYA command:

```
CHGQRYA JOB(123456/USERNAME/JOBNAME) QRYSTGLMT(100)
```

This sets the query temporary storage limit to 100 MBfor job 123456/USERNAME/JOBNAME.

Note: Unlike the query time limit, there is no system value for temporary storage limit. The default behavior is to let any queries run regardless of their temporary storage usage. The query temporary storage limit can be specified either in the INI file or on the **Change Query Attributes (CHGQRYA)** command.

Related information

Change Query Attributes (CHGQRYA) command

Controlling parallel processing for queries

There are two types of parallel processing available. The first is a parallel I/O that is available at no charge. The second is DB2 Symmetric Multiprocessing, a feature that you can purchase. You can turn parallel processing on and off.

Even if parallelism is enabled for a system or job, the individual queries that run in a job might not actually use a parallel method. This decision might be because of functional restrictions, or the optimizer might choose a non-parallel method because it runs faster.

Queries processed with parallel access methods aggressively use main storage, CPU, and disk resources. The number of queries that use parallel processing must be limited and controlled.

Controlling system-wide parallel processing for queries

You can use the QQRYDEGREE system value to control parallel processing for a system.

The current value of the system value can be displayed or modified using the following CL commands:

- **WRKSYSVAL** - **Work with System Value**
- **CHGSYSVAL** - **Change System Value**
- **DSPSYSVAL** - **Display System Value**
- **RTVSYSVAL** - **Retrieve System Value**

The special values for QQRYDEGREE control whether parallel processing is allowed by default for all jobs on the system. The possible values are:

***NONE**
: No parallel processing is allowed for database query processing.

***IO**
: I/O parallel processing is allowed for queries.

***OPTIMIZE**
: The query optimizer can choose to use any number of tasks for either I/O or SMP parallel processing to process the queries. SMP parallel processing is used only if the DB2 Symmetric Multiprocessing feature is installed. The query optimizer chooses to use parallel processing to minimize elapsed time based on the job share of the memory in the pool.

***MAX**
: The query optimizer can choose to use either I/O or SMP parallel processing to process the query. SMP parallel processing can be used only if the DB2 Symmetric Multiprocessing feature is installed. The choices made by the query optimizer are like the choices made for parameter value *OPTIMIZE. The exception is that the optimizer assumes that all active memory in the pool can be used to process the query.

The default QQRYDEGREE system value is *NONE. You must change the value if you want parallel query processing as the default for jobs run on the system.

Changing this system value affects all jobs that is run or are currently running on the system whose DEGREE query attribute is *SYSVAL. However, queries that have already been started or queries using reusable ODPs are not affected.

Controlling job level parallel processing for queries

You can also control query parallel processing at the job level using the DEGREE parameter of the **Change Query Attributes (CHGQRYA)** command or in the QAQQINI file. You can also use the SET_CURRENT_DEGREE SQL statement.

Using the Change Query Attributes (CHGQRYA) command

The parallel processing option allowed and, optionally, the number of tasks that can be used when running database queries in the job can be specified. You can prompt on the **Change Query Attributes (CHGQRYA)** command in an interactive job to display the current values of the DEGREE query attribute.

Changing the DEGREE query attribute does not affect queries that have already been started or queries using reusable ODPs.

The parameter values for the DEGREE keyword are:

<u>*SAME</u>
The parallel degree query attribute does not change.

***NONE**
No parallel processing is allowed for database query processing.

***IO**
Any number of tasks can be used when the database query optimizer chooses to use I/O parallel processing for queries. SMP parallel processing is not allowed.

***OPTIMIZE**
The query optimizer can choose to use any number of tasks for either I/O or SMP parallel processing to process the query. SMP parallel processing can be used only if the DB2 Symmetric Multiprocessing feature is installed. Use of parallel processing and the number of tasks used is determined by:

- the number of system processors available
- the job share of active memory available in the pool
- whether the expected elapsed time is limited by CPU processing or I/O resources

The query optimizer chooses an implementation that minimizes elapsed time based on the job share of the memory in the pool.

***MAX**
The query optimizer can choose to use either I/O or SMP parallel processing to process the query. SMP parallel processing can be used only if the DB2 Symmetric Multiprocessing feature is installed. The choices made by the query optimizer are like the choices made for parameter value *OPTIMIZE. The exception is that the optimizer assumes that all active memory in the pool can be used to process the query.

NBRTASKS** ***number-of-tasks
Specifies the number of tasks to be used when the query optimizer chooses to use SMP parallel processing to process a query. I/O parallelism is also allowed. SMP parallel processing can be used only if the DB2 Symmetric Multiprocessing feature is installed.

Using a number of tasks less than the number of system processors available restricts the number of processors used simultaneously for running a query. A larger number of tasks ensures that the query is allowed to use all the processors available on the system to run the query. Too many tasks can degrade performance because of the over commitment of active memory and the overhead cost of managing all the tasks.

***SYSVAL**
: Specifies that the processing option used is set to the current value of the QQRYDEGREE system value.

The initial value of the DEGREE attribute for a job is *SYSVAL.

Using the SET CURRENT DEGREE SQL statement

You can use the SET CURRENT DEGREE SQL statement to change the value of the CURRENT_DEGREE special register. The possible values for the CURRENT_DEGREE special register are:

1
: No parallel processing is allowed.

2 through 32767
: Specifies the degree of parallelism that is used.

ANY
: Specifies that the database manager can choose to use any number of tasks for either I/O or SMP parallel processing. Use of parallel processing and the number of tasks used is determined by:

- the number of system processors available
- the job share of active memory available in the pool
- whether the expected elapsed time is limited by CPU processing or I/O resources

The database manager chooses an implementation that minimizes elapsed time based on the job share of the memory in the pool.

NONE
: No parallel processing is allowed.

MAX
: The database manager can choose to use any number of tasks for either I/O or SMP parallel processing. MAX is like ANY except the database manager assumes that all active memory in the pool can be used.

IO
: Any number of tasks can be used when the database manager chooses to use I/O parallel processing for queries. SMP is not allowed.

The value can be changed by invoking the SET CURRENT DEGREE statement.

The initial value of CURRENT DEGREE comes from the CHGQRYA CL command, PARALLEL_DEGREE parameter in the current query options file (QAQQINI), or the QQRYDEGREE system value.

Related information

Set Current Degree statement
Change Query Attributes (CHGQRYA) command
DB2 Symmetric Multiprocessing

Collecting statistics with the statistics manager

The collection of statistics is handled by a separate component called the statistics manager. Statistical information can be used by the query optimizer to determine the best access plan for a query. Since the query optimizer bases its choice of access plan on the statistical information found in the table, it is important that this information is current.

On many platforms, statistics collection is a manual process that is the responsibility of the database administrator. With IBM i products, the database statistics collection process is handled automatically, and only rarely is it necessary to update statistics manually.

The statistics manager does not actually run or optimize the query. It controls the access to the metadata and other information that is required to optimize the query. It uses this information to answer questions posed by the query optimizer. The answers can either be derived from table header information, from existing indexes, or from single-column statistics.

The statistics manager must always provide an answer to the questions from the Optimizer. It uses the best method available to provide the answers. For example, it could use a single-column statistic or perform a key range estimate over an index. Along with the answer, the statistics manager returns a confidence level to the optimizer that the optimizer can use to provide greater latitude for sizing algorithms. If the statistics manager provides a low confidence in the number of groups estimated for a grouping request, the optimizer can increase the size of the temporary hash table allocated.

Related concepts

Statistics manager

In CQE, the retrieval of statistics is a function of the Optimizer. When the Optimizer needs to know information about a table, it looks at the table description to retrieve the row count and table size. If an index is available, the Optimizer might extract information about the data in the table. In SQE, the collection and management of statistics is handled by a separate component called the statistics manager. The statistics manager leverages all the same statistical sources as CQE, but adds more sources and capabilities.

Automatic statistics collection

When the statistics manager prepares its responses to the optimizer, it tracks the responses that were generated using default filter factors. Default filter factors are used when column statistics or indexes are not available. The statistics manager uses this information to automatically generate a statistic collection request for the columns. This request occurs while the access plan is written to the plan cache. If system resources allow, statistics collections occur in real time for direct use by the current query, avoiding a default answer to the optimizer.

Otherwise, as system resources become available, the requested column statistics are collected in the background. The next time the query is executed, the missing column statistics are available to the statistics manager. This process allows the statistics manager to provide more accurate information to the optimizer at that time. More statistics make it easier for the optimizer to generate a better performing access plan.

If a query is canceled before or during execution, the requests for column statistics are still processed. These requests occur if the execution reaches the point where the generated access plan is written to the Plan Cache.

To minimize the number of passes through a table during statistics collection, the statistics manger groups multiple requests for the same table. For example, two queries are executed against table T1. The first query has selection criteria on column C1 and the second over column C2. If no statistics are available for the table, the statistics manager identifies both of these columns as good candidates for column statistics. When the statistics manager reviews requests, it looks for multiple requests for the same table and groups them into one request. This grouping allows both column statistics to be created with only one pass through table T1.

One thing to note is that column statistics are usually automatically created when the statistics manager must answer questions from the optimizer using default filter factors. However, when an index is available that might be used to generate the answer, then column statistics are not automatically generated. In this scenario, there might be cases where optimization time benefits from column statistics. Using column statistics to answer questions from the optimizer is more efficient than using the index data. So if query performance seems extended, you might want to verify that there are indexes over the relevant columns in your query. If so, try manually generating column statistics for these columns.

As stated before, statistics collection occurs as system resources become available. If you have a low priority job permanently active on your system that is supposed to use all spare CPU cycles for processing, your statistics collection is never active.

Automatic statistics refresh

Column statistics are not maintained when the underlying table data changes. The statistics manager determines if columns statistics are still valid or if they no longer represent the column accurately (stale).

This validation is done each time one of the following occurs:

- A full open occurs for a query where column statistics were used to create the access plan

- A new plan is added to the plan cache, either because a new query was optimized or because an existing plan was reoptimized.

To validate the statistics, the statistics manager checks to see if any of the following apply:

- Number of rows in the table has changed by more than 15% of the total table row count
- Number of rows changed in the table is more than 15% of the total table row count

If the statistics are stale, the statistics manager still uses them to answer the questions from the optimizer. However, the statistics manager marks the statistics as stale in the plan cache and generates a request to refresh them.

Viewing statistics requests
You can view the current statistics requests by using System i Navigator or by using Statistics APIs.

To view requests in System i Navigator, right-click **Database** and select **Statistic Requests**. This window shows all user requested statistics collections that are pending or active. The view also shows all system requested statistics collections that are being considered, are active, or have failed. You can change the status of the request, order the request to process immediately, or cancel the request.

Related reference

Statistics manager APIs
You can use APIs to implement the statistics function of System i Navigator.

Indexes and column statistics
While performing similar functions, indexes and column statistics are different.

If you are trying to decide whether to use statistics or indexes to provide information to the statistics manager, keep in mind the following differences.

One major difference between indexes and column statistics is that indexes are permanent objects that are updated when changes to the underlying table occur. Column statistics are not updated. If your data is constantly changing, the statistics manager might need to rely on stale column statistics. However, maintaining an index after each table change might use more system resources than refreshing stale column statistics after a group of changes have occurred.

Another difference is the effect that the existence of new indexes or column statistics has on the optimizer. When new indexes become available, the optimizer considers them for implementation. If they are candidates, the optimizer reoptimizes the query and tries to find a better implementation. However, this reoptimization is not true for column statistics. When new or refreshed column statistics are available, the statistics manager interrogates immediately. Reoptimization occurs only if the answers are different from the ones that were given before these refreshed statistics. It is possible to use statistics that are refreshed without causing a reoptimization of an access plan.

When trying to determine the selectivity of predicates, the statistics manager considers column statistics and indexes as resources for its answers in the following order:

1. Try to use a multi-column keyed index when ANDed or ORed predicates reference multiple columns
2. If there is no perfect index that contains all the columns in the predicates, it tries to find a combination of indexes that can be used.
3. For single column questions, it uses available column statistics
4. If the answer derived from the column statistics shows a selectivity of less than 2%, indexes are used to verify this answer

Accessing column statistics to answer questions is faster than trying to obtain these answers from indexes.

Column statistics can only be used by SQE. For CQE, all statistics are retrieved from indexes.

Finally, column statistics can be used only for query optimization. They cannot be used for the actual implementation of a query, whereas indexes can be used for both.

Monitoring background statistics collection

The system value QDBFSTCCOL controls who is allowed to create statistics in the background.

The following list provides the possible values:

***ALL**
Allows all statistics to be collected in the background. *ALL is the default setting.

***NONE**
Restricts everyone from creating statistics in the background. *NONE does not prevent immediate user-requested statistics from being collected, however.

***USER**
Allows only user-requested statistics to be collected in the background.

***SYSTEM**
Allows only system-requested statistics to be collected in the background.

When you switch the system value to something other than *ALL or *SYSTEM, the statistics manager continues to place statistics requests in the plan cache. When the system value is switched back to *ALL, for example, background processing analyzes the entire plan cache and looks for any existing column statistics requests. This background task also identifies column statistics that have been used by a plan in the plan cache. The task determines if these column statistics have become stale. Requests for the new column statistics as well as requests for refresh of the stale columns statistics are then executed.

All background statistic collections initiated by the system or submitted by a user are performed by the system job QDBFSTCCOL. User-initiated immediate requests are run within the user job. This job uses multiple threads to create the statistics. The number of threads is determined by the number of processors that the system has. Each thread is then associated with a request queue.

There are four types of request queues based on who submitted the request and how long the collection is estimated to take. The default priority assigned to each thread can determine to which queue the thread belongs:

- Priority 90 — short user requests
- Priority 93 — long user requests
- Priority 96 — short system requests
- Priority 99 — long system requests

Background statistics collections attempt to use as much parallelism as possible. This parallelism is independent of the SMP feature installed on the system. However, parallel processing is allowed only for immediate statistics collection if SMP is installed on the system. The job that requests the column statistics also must allow parallelism.

Related information

Performance system values: Allow background database statistics collection

Replication of column statistics with CRTDUPOBJ versus CPYF

You can replicate column statistics with the **`Create Duplicate Object (CRTDUPOBJ)`** or the **`Copy File (CPYF)`** commands.

Statistics are not copied to new tables when using the **`Copy File (CPYF)`** command. If statistics are needed immediately after using this command, then you must manually generate the statistics using System i Navigator or the statistics APIs. If statistics are not needed immediately, then they could be created automatically by the system after the first touch of a column by a query.

Statistics are copied when using **`Create Duplicate Object (CRTDUPOBJ)`** command with DATA(*YES). You can use this command as an alternative to creating statistics automatically after using a **`Copy File (CPYF)`** command.

Related information

Create Duplicate Object (CRTDUPOBJ) command

Copy File (CPYF) command

Determining what column statistics exist

You can determine what column statistics exist in a couple of ways.

The first is to view statistics by using System i Navigator. Right-click a table or alias and select **Statistic Data**. Another way is to create a user-defined table function and call that function from an SQL statement or stored procedure.

Manually collecting and refreshing statistics

You can manually collect and refresh statistics through System i Navigator or by using statistics APIs.

To collect statistics using System i Navigator, right-click a table or alias and select Statistic Data. On the Statistic Data dialog, click New. Then select the columns that you want to collect statistics for. Once you have selected the columns, you can collect the statistics immediately or collect them in the background.

To refresh a statistic using System i Navigator, right-click a table or alias and select **Statistic Data**. Click **Update**. Select the statistic that you want to refresh. You can collect the statistics immediately or collect them in the background.

There are several scenarios in which the manual management (create, remove, refresh, and so on) of column statistics could be beneficial and recommended.

High Availability (HA) solutions

High availability solutions replicate data to a secondary system by using journal entries. However, column statistics are not journaled. That means that, on your backup system, no column statistics are available when you first start using that system. To prevent the "warm up" effect, you might want to propagate the column statistics that were gathered on your production system. Recreate them on your backup system manually.

ISV (Independent Solution Provider) preparation

An ISV might want to deliver a customer solution that already includes column statistics frequently used in the application, rather than waiting for the automatic statistics collection to create them. Run the application on the development system for some time and examine which column statistics were created automatically. You can then generate a script file to execute on the customer system after the initial data load takes place. The script file can be shipped as part of the application

Business Intelligence environments

In a large Business Intelligence environment, it is common for large data load and update operations to occur overnight. Column statistics are marked stale only when they are touched by the statistics manager, and then refreshed after first touch. You might want to consider refreshing the column statistics manually after loading the data.

You can do this refresh easily by toggling the system value QDBFSTCCOL to *NONE and then back to *ALL. This process causes all stale column statistics to be refreshed. It also starts collection of any column statistics previously requested by the system but not yet available. Since this process relies on the access plans stored in the plan cache, avoid performing a system initial program load (IPL) before toggling QDBFSTCCOL. An IPL clears the plan cache.

This procedure works only if you do not delete (drop) the tables and recreate them in the process of loading your data. When deleting a table, access plans in the plan cache that refer to this table are deleted. Information about column statistics on that table is also lost. The process in this environment is either to add data to your tables or to clear the tables instead of deleting them.

Massive data updates

Updating rows in a column statistics-enabled table can significantly change the cardinality, add new ranges of values, or change the distribution of data values. These updates can affect query performance on the first query run against the new data. On the first run of such a query, the optimizer uses stale column statistics to determine the access plan. At that point, it starts a request to refresh the column statistics.

Prior to this data update, you might want to toggle the system value QDBFSTCCOL to *NONE and back to *ALL or *SYSTEM. This toggle causes an analysis of the plan cache. The analysis includes searching for column statistics used in access plan generation, analyzing them for staleness, and requesting updates for the stale statistics.

If you massively update or load data, and run queries against these tables at the same time, the automatic column statistics collection tries to refresh every time 15% of the data is changed. This processing can be redundant since you are still updating or loading the data. In this case, you might want to block automatic statistics collection for the tables and deblock it again after the data update or load finishes. An alternative is to turn off automatic statistics collection for the whole system before updating or loading the data. Switch it back on after the updating or loading has finished.

Backup and recovery
When thinking about backup and recovery strategies, keep in mind that creation of column statistics is not journaled. Column statistics that exist at the time a save operation occurs are saved as part of the table and restored with the table. Any column statistics created after the save took place are lost and cannot be recreated by using techniques such as applying journal entries. If you have a long interval between save operations and rely on journaling to restore your environment, consider tracking column statistics that are generated after the latest save operation.

Related information

Performance system values: Allow background database statistics collection

Statistics manager APIs

You can use APIs to implement the statistics function of System i Navigator.

- `Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics)` immediately cancels statistics collections that have been requested, but are not yet completed or not successfully completed.
- `Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics)` immediately deletes existing completed statistics collections.
- `List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics)` lists all the columns and combination of columns and file members that have background statistic collections requested, but not yet completed.
- `List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics)` lists additional statistics data for a single statistics collection.
- `List Statistics Collections (QDBSTLS, QdbstListStatistics)` lists all the columns and combination of columns for a given file member that have statistics available.
- `Request Statistics Collections (QDBSTRS, QdbstRequestStatistics)` allows you to request one or more statistics collections for a given set of columns of a specific file member.
- `Update Statistics Collection (QDBSTUS, QdbstUpdateStatistics)` allows you to update the attributes and to refresh the data of an existing single statistics collection

Related reference

Viewing statistics requests
You can view the current statistics requests by using System i Navigator or by using Statistics APIs.

Displaying materialized query table columns

You can display materialized query tables associated with another table using System i Navigator.

To display materialized query tables, follow these steps:

1. In the System i Navigator window, expand the system that you want to use.
2. Expand **Databases** and the database that you want to work with.
3. Expand **Schemas** and the schema that you want to work with.
4. Right-click a table and select **Show Materialized Query Tables**.

Table 48. Columns used in Show materialized query table window

Column name	Description
Name	The SQL name for the materialized query table

Table 48. Columns used in Show materialized query table window (continued)

Column name	Description
Schema	Schema or library containing the materialized query table
Partition	Partition detail for the index. Possible values: • <blank>, which means For all partitions • For Each Partition • specific name of the partition
Owner	The user ID of the owner of the materialized query table.
System Name	System table name for the materialized query table
Enabled	Whether the materialized query table is enabled. Possible values are: • Yes • No If the materialized query table is not enabled, it cannot be used for query optimization. It can, however, be queried directly.
Creation Date	The timestamp of when the materialized table was created.
Last Refresh Date	The timestamp of the last time the materialized query table was refreshed.
Last Query Use	The timestamp when the materialized query table was last used by the optimizer to replace user specified tables in a query.
Last Query Statistics Use	The timestamp when the materialized query table was last used by the statistics manager to determine an access method.
Query Use Count	The number of instances the materialized query table was used by the optimizer to replace user specified tables in a query.
Query Statistics Use Count	The number of instances the materialized query table was used by the statistics manager to determine an access method.
Last Used Date	The timestamp when the materialized query table was last used.
Days Used Count	The number of days the materialized query table has been used.
Date Reset Days Used Count	The year and date when the days-used count was last set to 0.
Current Number of Rows	The total number of rows included in this materialized query table at this time.
Current Size	The current size of the materialized query table.
Last Changed	The timestamp when the materialized query table was last changed.

Table 48. Columns used in Show materialized query table window (continued)	
Column name	**Description**
Maintenance	The maintenance for the materialized query table. Possible values are: • User • System
Initial Data	Whether the initial data was inserted immediately or deferred. Possible values are • Deferred • Immediate
Refresh Mode	The refresh mode for the materialized query table. A materialized query table can be refreshed whenever a change is made to the table or deferred to a later time.
Isolation Level	The isolation level for the materialized query table.
Sort Sequence	The alternate character sorting sequence for National Language Support (NLS).
Language Identifier	The language code for the object.
SQL Statement	The SQL statement that is used to populate the table.
Text	The text description of the materialized query table.
Table	Schema and table name.
Table Partition	Table partition.
Table System Name	System name of the table.

Managing check pending constraints columns

You can view and change constraints that have been placed in a check pending state by the system. Check pending constraints refers to a state in which a mismatch exists between a parent and foreign key in a referential constraint. A mismatch can also occur between the column value and the check constraint definition in a check constraint.

To view constraints that have been placed in a check pending state, follow these steps:

1. Expand the system name and **Databases**.
2. Expand the database that you want to work with.
3. Expand the **Database Maintenance** folder.
4. Select **Check Pending Constraints**.
5. From this interface, you can view the definition of the constraint and the rows that are in violation of the constraint rules. Select the constraint that you want to work with and then select **Edit Check Pending Constraint** from the **File** menu.
6. You can either alter or delete the rows that are in violation.

Table 49. Columns used in Check pending constraints window	
Column name	**Description**
Name of Constraint in Check Pending	Displays the name of the constraint that is in a check pending state.

Table 49. Columns used in Check pending constraints window (continued)	
Column name	**Description**
Schema	Schema containing the constraint that is in a check pending state.
Type	Displays the type of constraint that is in check pending. Possible values are: Check constraint Foreign key constraint
Table name	The name of the table associated with the constraint in check pending state.
Enabled	Displays whether the constraint is enabled. The constraint must be disabled or the relationship taken out of the check pending state before any input/output (I/O) operations can be performed.

Creating an index strategy

Db2 for i provides two basic means for accessing tables: a table scan and an index-based retrieval. Index-based retrieval is typically more efficient than table scan when less than 20% of the table rows are selected.

There are two kinds of persistent indexes: binary radix tree indexes, which have been available since 1988, and encoded vector indexes (EVIs), which became available in 1998 with V4R2. Both types of indexes are useful in improving performance for certain kinds of queries.

Binary radix indexes

A radix index is a multilevel, hybrid tree structure that allows many key values to be stored efficiently while minimizing access times. A key compression algorithm assists in this process. The lowest level of the tree contains the leaf nodes, which contain the base table row addresses associated with the key value. The key value is used to quickly navigate to the leaf node with a few simple binary search tests.

The binary radix tree structure is good for finding a few rows because it finds a given row with a minimal amount of processing. For example, create a binary radix index over a customer number column. Then create a typical OLTP request like "find the outstanding orders for a single customer". The binary index results in fast performance. An index created over the customer number column is considered the perfect index for this type of query. The index allows the database to find the rows it needs and perform a minimal number of I/Os.

In some situations, however, you do not always have the same level of predictability. Many users want on demand access to the detail data. For example, they might run a report every week to look at sales data. Then they want to "drill down" for more information related to a particular problem area they found in the report. In this scenario, you cannot write all the queries in advance on behalf of the end users. Without knowing what queries might run, it is impossible to build the perfect index.

Related information

SQL Create Index statement

Derived key index

You can use the SQL CREATE INDEX statement to create a derived key index using an SQL expression.

Traditionally an index could only specify column names in the key of the index over the table it was based on. With this support, an index can have an expression in place of a column name that can use built-in functions, or some other valid expression. Additionally, you can use the SQL CREATE INDEX statement to create a sparse index using a WHERE condition.

For restrictions and other information about derived indexes, see the Create Index statement and Using derived indexes.

Related reference

Using derived indexes
SQL indexes can be created where the key is specified as an expression. This type of key is also referred to as a derived key.

Related information

SQL Create Index statement

Sparse indexes

You can use the SQL CREATE INDEX statement to create a sparse index using SQL selection predicates.

Last release users were given the ability to use the SQL CREATE INDEX statement to create a sparse index using a WHERE condition. With this support, the query optimizer recognizes and considers sparse indexes during its optimization. If the query WHERE selection is a subset of the sparse index WHERE selection, then the sparse index is used to implement the query. Use of the sparse index usually results in improved performance.

Examples

In this example, the query selection is a subset of the sparse index selection and an index scan over the sparse index is used. The remaining query selection (COL3=30) is executed following the index scan.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In this example, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20
```

Related reference

Using sparse indexes
SQL indexes can be created using WHERE selection predicates. These indexes can also be referred to as sparse indexes. The advantage of a sparse index is that fewer entries are maintained in the index. Only those entries matching the WHERE selection criteria are maintained in the index.

Related information

SQL Create Index statement

Sparse index optimization

An SQL sparse index is like a select/omit access path. Both the sparse index and the select/omit logical file contain only keys that meet the selection specified. For a sparse index, the selection is specified with a WHERE clause. For a select/omit logical file, the selection is specified in the DDS using the COMP operation.

The reason for creating a sparse index is to provide performance enhancements for your queries. The performance enhancement is done by precomputing and storing results of the WHERE selection in the sparse index. The database engine can use these results instead of recomputing them for a user specified query. The query optimizer looks for any applicable sparse index and can choose to implement the query using a sparse index. The decision is based on whether using a sparse index is a faster implementation choice.

For a sparse index to be used, the WHERE selection in the query must be a subset of the WHERE selection in the sparse index. That is, the set of records in the sparse index must contain all the records to be selected by the query. It might contain extra records, but it must contain all the records to be selected by the query. This comparison of WHERE selection is performed by the query optimizer during optimization. It is like the comparison that is performed for Materialized Query Tables (MQT).

Besides the comparison of the WHERE selection, the optimization of a sparse index is identical to the optimization that is performed for any Binary Radix index.

Refer to section 'Indexes and the Optimizer' for more details on how Binary Radix indexes are optimized.

Related concepts

Indexes & the optimizer
Since the optimizer uses cost based optimization, more information about the database rows and columns makes for a more efficient access plan created for the query. With the information from the indexes, the optimizer can make better choices about how to process the request (local selection, joins, grouping, and ordering).

Related reference

Using sparse indexes
SQL indexes can be created using WHERE selection predicates. These indexes can also be referred to as sparse indexes. The advantage of a sparse index is that fewer entries are maintained in the index. Only those entries matching the WHERE selection criteria are maintained in the index.

Sparse index matching algorithm

This topic is a generalized discussion of how the sparse index matching algorithm works.

The selection in the query must be a subset of the selection in the sparse index in order for the sparse index to be used. This statement is true whether the selection is ANDed together, ORed together, or a combination of the two. For selection where all predicates are ANDed together, all WHERE selection predicates specified in the sparse index must also be specified in the query. The query can contain additional ANDed predicates. The selection for the additional predicates will be performed after the entries are retrieved from the sparse index. See examples A1, A2, and A3 following.

Example A1

In this example, the query selection exactly matches the sparse index selection and an index scan over the sparse index can be used.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

Example A2

In this example, the query selection is a subset of the sparse index selection and an index scan over the sparse index can be used. The remaining query selection (COL3=30) is executed following the index scan.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

Example A3

In this example, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20
```

For selection where all predicates are ORed together, all WHERE selection predicates specified in the query, must also be specified in the sparse index. The sparse index can contain additional ORed predicates. All the ORed selection in the query will be executed after the entries are retrieved from the sparse index. See examples O1, O2, andO3 following.

Example O1

In this example, the query selection exactly matches the sparse index selection and an index scan over the sparse index can be used. The query selection is executed following the index scan.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20 or COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20 or COL3=30
```

Example O2

In this example, the query selection is a subset of the sparse index selection and an index scan over the sparse index can be used. The query selection is executed following the index scan.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20 or COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20
```

Example O3

In this example, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20 or COL3=30
```

The previous examples used simple selection, all ANDed, or all ORed together. These examples are not typical, but they demonstrate how the selection of the sparse index is compared to the selection of the query. Obviously, the more complex the selection the more difficult it becomes to determine compatibility.

In the next example T1, the constant 'MN' was replaced by a parameter marker for the query selection. The sparse index had the local selection of COL1='MN' applied to it when it was created. The sparse index matching algorithm matches the parameter marker to the constant 'MN' in the query predicate COL1 =?. It verifies that the value of the parameter marker is the same as the constant in the sparse index; therefore the sparse index can be used.

The sparse index matching algorithm attempts to match where the predicates between the sparse index and the query are not the same. An example is a sparse index with a predicate SALARY > 50000, and a query with the predicate SALARY > 70000. The sparse index contains the rows necessary to run the query. The sparse index is used in the query, but the predicate SALARY > 70000 remains as selection in the query (it is not removed).

Example T1

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=? or COL2='TWINS' or COL3='WIN'
```

In the next example T2, the keys of the sparse index match the ORDER BY fields in the query. For the sparse index to satisfy the specified ordering, the optimizer must verify that the query selection is a subset of the sparse index selection. In this example, the sparse index can be used.

Example T2

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL1, COL3)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE COL2='TWINS'
ORDER  BY COL1, COL3
```

Related reference

Using sparse indexes
SQL indexes can be created using WHERE selection predicates. These indexes can also be referred to as sparse indexes. The advantage of a sparse index is that fewer entries are maintained in the index. Only those entries matching the WHERE selection criteria are maintained in the index.

Details on the MQT matching algorithm
What follows is a generalized discussion of how the MQT matching algorithm works.

Sparse index examples

This topic shows examples of how the sparse index matching algorithm works.

In example S1, the query selection is a subset of the sparse index selection and consequently an index scan over the sparse index is used. The remaining query selection (COL3=30) is executed following the index scan.

Example S1

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In example S2, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

Example S2

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20
```

In example S3, the query selection exactly matches the sparse index selection and an index scan over the sparse index can be used.

Example S3

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In example S4, the query selection is a subset of the sparse index selection and an index scan over the sparse index can be used. The remaining query selection (COL3=30) is executed following the index scan.

Example S4

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In example S5, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

Example S5

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20
```

In example S6, the query selection exactly matches the sparse index selection and an index scan over the sparse index can be used. The query selection is executed following the index scan to eliminate excess records from the sparse index.

Example S6

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20 or COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20 or COL3=30
```

In example S7, the query selection is a subset of the sparse index selection and an index scan over the sparse index can be used. The query selection is executed following the index scan to eliminate excess records from the sparse index.

Example S7

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20 or COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20
```

In example S8, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

Example S8

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20 or COL3=30
```

In the next example S9, the constant 'MN' was replaced by a parameter marker for the query selection. The sparse index had the local selection of COL1='MN' applied to it when it was created. The sparse index matching algorithm matches the parameter marker to the constant 'MN' in the query predicate COL1 =?. It verifies that the value of the parameter marker is the same as the constant in the sparse index; therefore the sparse index can be used.

Example S9

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
Where Col3='WIN' and (Col1=? or Col2='TWINS')
```

In the next example S10, the keys of the sparse index match the order by fields in the query. For the sparse index to satisfy the specified ordering, the optimizer must verify that the query selection is a subset of the sparse index selection. In this example, the sparse index can be used.

Example S10

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL1, COL3)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
Where Col3='WIN' and (Col1='MN' or Col2='TWINS')
ORDER  BY COL1, COL3
```

In the next example S11, the keys of the sparse index do not match the order by fields in the query. But the selection in sparse index T2 is a superset of the query selection. Depending on size, the optimizer might choose an index scan over sparse index T2 and then use a sort to satisfy the specified ordering.

Example S11

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL2, COL4)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
Where Col3='WIN' and (Col1='MN' or Col2='TWINS')
ORDER  BY COL1, COL3
```

The next example S12 represents the classic optimizer decision: is it better to do an index probe using index IX1 or is it better to do an index scan using sparse index SPR1? Both indexes retrieve the same number of index entries and have the same cost from that point forward. For example, both indexes have the same cost to retrieve the selected records from the dataspace, based on the retrieved entries/keys.

The cost to retrieve the index entries is the deciding criteria. In general, if index IX1 is large then an index scan over sparse index SPR1 has a lower cost to retrieve the index entries. If index IX1 is rather small then an index probe over index IX1 has a lower cost to retrieve the index entries. Another cost decision is reusability. The plan using sparse index SPR1 is not as reusable as the plan using index IX1 because of the static selection built into the sparse selection.

Example S12

```
CREATE INDEX MYLIB/IX1 on MYLIB/T1 (COL1, COL2, COL3)
```

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
CSELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

Specify PAGESIZE on index creates

You can use the PAGESIZE parameter to specify the access path logical page size used by the system when the access path is created. Use the PAGESIZE parameter when creating keyed files or indexes using the **Create Physical File (CRTPF)** or **Create Logical File (CRTLF)** commands, or the SQL CREATE INDEX statement.

The logical page size is the access path number of bytes that can be moved from auxiliary storage to the job storage pool for a page fault.

Consider using the default of *KEYLEN for this parameter, except in rare circumstances. Then the page size can be determined by the system based on the total length of the keys. When the access path is used by selective queries (for example, individual key lookup), a smaller page size is typically more efficient. When the query-selected keys are grouped in the access path with many records selected, or the access path is scanned, a larger page size is more efficient.

Related information

Create Logical File (CRTLF) command

Create Physical File (CRTPF) command

SQL Create Index statement

General index maintenance

Whenever indexes are created and used, there is a potential for a decrease in I/O velocity due to maintenance. Therefore, consider the maintenance cost of creating and using additional indexes. For radix indexes with MAINT(*IMMED), maintenance occurs when inserting, updating, or deleting rows.

To reduce the maintenance of your indexes consider:

- Minimizing the number of table indexes by creating composite (multiple column) key indexes. Composite indexes can be used for multiple different situations.
- Dropping indexes during batch inserts, updates, and deletes
- Creating in parallel. Either create indexes, one at a time, in parallel using SMP or create multiple indexes simultaneously with multiple batch jobs
- Maintaining indexes in parallel using SMP

The goal of creating indexes is to improve query performance by providing statistics and implementation choices. Maintain a reasonable balance on the number of indexes to limit maintenance overhead.

Encoded vector indexes

An encoded vector index (EVI) is used to provide fast data access in decision support and query reporting environments.

EVIs are a complementary alternative to existing index objects (binary radix tree structure - logical file or SQL index) and are a variation on bitmap indexing. Because of their compact size and relative simplicity, EVIs provide for faster scans of a table that can also be processed in parallel.

An EVI is a data structure that is stored as two components:

- The symbol table contains statistical and descriptive information about each distinct key value represented in the table. Each distinct key is assigned a unique code, either 1 byte, 2 bytes or 4 bytes in size.

  By specifying INCLUDE on the create, additional aggregate values can be maintained in real time as an extension of the key portion of the symbol table entry. These aggregated values are over non-key data in the table grouped by the specified EVI key.
- The vector is an array of codes listed in the same ordinal position as the rows in the table. The vector does not contain any pointers to the actual rows in the table.

Advantages of EVIs:

- Require less storage
- May have better build times than radix, especially if the number of unique values in the columns defined for the key is relatively small.
- Provide more accurate statistics to the query optimizer
- Considerably better performance for certain grouping types of queries
- Good performance characteristics for decision support environments.
- Can be further extended for certain types of grouping queries with the addition of INCLUDE values. Provides ready-made numeric aggregate values maintained in real time as part of index maintenance. INCLUDE values become an extension of the EVI symbol table. Multiple include values can be specified over different aggregating columns and maintained in the same EVI provided the group by values are the same. This technique can reduce overall maintenance.

Disadvantages of EVIs:

- Cannot be used in ordering.
- Use for grouping is specialized. Supports:
  - COUNT, DISTINCT requests over key columns
  - aggregate requests over key columns where all other selection can be applied to the EVI symbol table keys
  - INCLUDE aggregates
  - MIN or MAX, if aggregating value is part of the symbol table key.
- Use with joins always done in cooperation with hash table processing.
- Some additional maintenance idiosyncrasies.

Related reference

Encoded vector index
An encoded vector index is a permanent object that provides access to a table. This access is done by assigning codes to distinct key values and then representing those values in a vector.

Related information

SQL Create Index statement

SQL INCLUDE statement

How the EVI works
EVIs work in different ways for costing and implementation.

For costing, the optimizer uses the symbol table to collect metadata information about the query.

For implementation, the optimizer can use the EVI in one of the following ways:

- **Selection (WHERE clause)**

  The database engine uses the vector to build a dynamic bitmap or list of selected row ids. The bitmap or list contains 1 bit for each row in the table. The bit is turned on for each selected row. Like a bitmap index, these intermediate dynamic bitmaps (or lists) can be ANDed and ORed together to satisfy a query.

  For example, a user wants to see sales data for a specific region and time period. You can define an EVI over the `region` and `quarter` columns of the table. When the query runs, the database engine builds dynamic bitmaps using the two EVIs. The bitmaps are ANDed together to produce a single bitmap containing only the relevant rows for both selection criteria.

  This ANDing capability drastically reduces the number of rows that the system must read and test. The dynamic bitmaps exists only as long as the query is executing. Once the query is completed, the dynamic bitmaps are eliminated.

- **Grouping or Distinct**

The symbol table within the EVI contains distinct values for the specified columns in the key definition. The symbol table also contains a count of the number of records in the base table that have each distinct value. Queries involving grouping or distinct, based solely on columns in the key, are candidates for a technique that uses the symbol table directly to determine the query result.

The symbol table contains only the key values and their associated counts, unless INCLUDE is specified. Therefore, queries involving column function COUNT are eligible for this technique. But queries with column functions MIN or MAX on other non-key columns are not eligible. MIN and MAX values are not stored in the symbol table.

- **EVI INCLUDE aggregates**

  Including additional aggregate values further extends the ability of the symbol table to provide ready-made results. Aggregate data is grouped by the specified columns in the key definition. Therefore, aggregate data must be over columns in the table other than those columns specified as EVI key values.

  For performance, these included aggregates are limited to numeric results (SUM, COUNT, AVG, VARIANCE) as they can be maintained directly from the inserted or removed row.

  MIN or MAX values would occasionally require other row comparisons during maintenance and therefore are not supported with the INCLUDE keyword.

  EVI symbol table only access is used to satisfy distinct or grouping requests when the query is run with commitment control *NONE or *CHG.

  INCLUDE for additional aggregate values can be used in join queries. When possible, the existence of EVIs with INCLUDE aggregates causes the group by process to be pushed down to each table as necessary. See the following EVI INCLUDE grouping push down example: "EVI INCLUDE aggregate example" on page 73

Related reference

Encoded vector index index-only access
The encoded vector index can also be used for index-only access.

Encoded vector index symbol table scan
An encoded vector index symbol table scan operation is used to retrieve the entries from the symbol table portion of the index.

Encoded vector index symbol table probe
An encoded vector index symbol table probe operation is used to retrieve entries from the symbol table portion of the index. Scanning the entire symbol table is not necessary.

Index grouping implementation
There are two primary ways to implement grouping using an index: Ordered grouping and pre-summarized processing.

Related information

SQL INCLUDE statement

When to create EVIs

There are several instances to consider creating EVIs.

Consider creating encoded vector indexes when any one of the following is true:

- You want to gather 'live' statistics
- Full table scan is currently being selected for the query
- Selectivity of the query is 20%-70% and using skip sequential access with dynamic bitmaps speed up the scan
- When a star schema join is expected to be used for star schema join queries.
- When grouping or distinct queries are specified against a column, the columns have few distinct values and only the COUNT column function, if any, is used.
- When ready-made aggregate results grouped by the specified key columns would benefit query performance.

Create encoded vector indexes with:

- Single key columns with a low number of distinct values expected
- Keys columns with a low volatility (do not change often)
- Maximum number of distinct values expected using the WITH n DISTINCT VALUES clause
- Single key over foreign key columns for a star schema model

EVI with INCLUDE vs Materialized Query Tables

Although EVIs with INCLUDE are not a substitute for Materialized Query Tables (MQTs), INCLUDE EVIs have an advantage over single table aggregate MQTs (materialized query tables). The advantage is that the ready-made aggregate results are maintained in real time, not requiring explicit REFRESH TABLE requests. For performance and read access to aggregate results, consider turning your single table, aggregate MQTs into INCLUDE EVIs. Keep in mind that the other characteristics of a good EVI are applicable, such as a relatively low number of distinct key values.

As indexes, these EVIs are found during optimization just as any other indexes are found. Unlike MQTs, there is no INI setting to enable and no second pass through the optimizer to cost the application of this form of ready-made aggregate. In addition, EVIs with INCLUDE can be used to populate MQT summary tables if the EVI is a match for a portion of the MQT definition.

Related reference

Encoded vector index symbol table scan
An encoded vector index symbol table scan operation is used to retrieve the entries from the symbol table portion of the index.

Index grouping implementation
There are two primary ways to implement grouping using an index: Ordered grouping and pre-summarized processing.

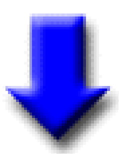
Related information

SQL INCLUDE statement

EVI maintenance

There are unique challenges to maintaining EVIs. The following table shows a progression of how EVIs are maintained, the conditions under which EVIs are most effective, and where EVIs are least effective, based on the EVI maintenance characteristics.

Table 50. EVI Maintenance Considerations

	Condition	Characteristics
Most Effective	When inserting an existing distinct key value	• Minimum overhead • Symbol table key value looked up and statistics updated • Vector element added for new row, with existing byte code • Minimal additional pathlength to maintain any INCLUDEd aggregate values (the increment of a COUNT or adding to an accumulating SUM)
	When inserting a *new* distinct key value - in order, within byte code range	• Minimum overhead • Symbol table key value added, byte code assigned, statistics assigned • Vector element added for new row, with new byte code • Minimal additional pathlength to maintain any INCLUDEd aggregate values (the increment of a COUNT or adding to an accumulating SUM)
	When inserting a new distinct key value - out of order, within byte code range	• Minimum overhead if contained within overflow area threshold • Symbol table key value added to overflow area, byte code assigned, statistics assigned • Vector element added for new row, with new byte code • Considerable overhead if overflow area threshold reached • Access path validated - not available • EVI refreshed, overflow area keys incorporated, new byte codes assigned (symbol table and vector elements updated) • Minimal additional path-length to maintain any INCLUDEd aggregate values (the increment of a COUNT or adding to an accumulating SUM)
	When inserting a new distinct key value - out of byte code range	• Considerable overhead • Access plan invalidated - not available • EVI refreshed, next byte code size used, new byte codes assigned (symbol table and vector elements updated • Not applicable to EVIs with INCLUDE, as by definition the

Related reference

Encoded vector index

An encoded vector index is a permanent object that provides access to a table. This access is done by assigning codes to distinct key values and then representing those values in a vector.

Related information

SQL INCLUDE statement

Recommendations for EVI use

Encoded vector indexes are a powerful tool for providing fast data access in decision support and query reporting environments. To ensure the effective use of EVIs, use the following guidelines.

Create EVIs on

- Read-only tables or tables with a minimum of INSERT, UPDATE, DELETE activity.
- Key columns that are used in the WHERE clause - local selection predicates of SQL requests.
- Single key columns that have a relatively small set of distinct values.
- Multiple key columns that result in a relatively small set of distinct values.
- Key columns that have a static or relatively static set of distinct values.
- Non-unique key columns, with many duplicates.

Create EVIs with the maximum byte code size expected

- Use the "WITH n DISTINCT VALUES" clause on the CREATE ENCODED VECTOR INDEX statement.
- If unsure, use a number greater than 65,535 to create a 4 byte code. This method avoids the EVI maintenance involved in switching byte code sizes.
- EVIs with INCLUDE always create with a 4 byte code.

When loading data

- Drop EVIs, load data, create EVIs.
- EVI byte code size is assigned automatically based on the number of actual distinct key values found in the table.
- Symbol table contains all key values, in order, no keys in overflow area.
- EVIs with INCLUDE always use 4 byte code

Consider adding INCLUDE values to existing EVIs

An EVI index with INCLUDE values can be used to supply ready-made aggregate results. The existing symbol table and vector are still used for table selection, when appropriate, for skip sequential plans over large tables, or for index ANDing and ORing plans. If you already have EVIs, consider creating new ones with additional INCLUDE values, and then drop the pre-existing index.

Consider specifying multiple INCLUDE values on the same EVI create

If you need different aggregates over different table values for the same GROUP BY columns specified as EVI keys, define those aggregates in the same EVI. This definition cuts down on maintenance costs and allows for a single symbol table and vector.

For example:

```
Select SUM(revenue) from sales group by Country
```

```
Select SUM(costOfGoods) from sales group by Country, Region
```

Both queries could benefit from the following EVI:

```
CREATE ENCODED VECTOR INDEX  eviCountryRegion on Sales(country,region)
       INCLUDE(SUM(revenue), SUM(costOfGoods))
```

The optimizer does additional grouping (regrouping) if the EVI key values are wider than the corresponding GROUP BY request of the query. This additional grouping would be the case in the first example query.

If an aggregate request is specified over null capable results, an implicit COUNT over that same result is included as part of the symbol table entry. The COUNT is used to facilitate index maintenance when a requested aggregate needs to reflect. It can also assist with pushing aggregation through a join if the optimizer determines this push is possible. The COUNT is then used to help compensate for fewer join activity due to the pushed down grouping.

Consider EVI INCLUDE and Grouping Sets

EVI INCLUDE support has been expanded to match GROUPING SETs, ROLLUP and CUBE queries.

When EVI INCLUDES are available over a table being aggregated over in a grouping sets query, the query is rewritten to facilitate and match any EVI INCLUDE indexes that might be available. This can result in exceeding good query performance because the table is never accessed. All the aggregate variations necessary to perform the rollup, cube or grouping set query result can be performed over the EVI symbol table with INCLUDE values.

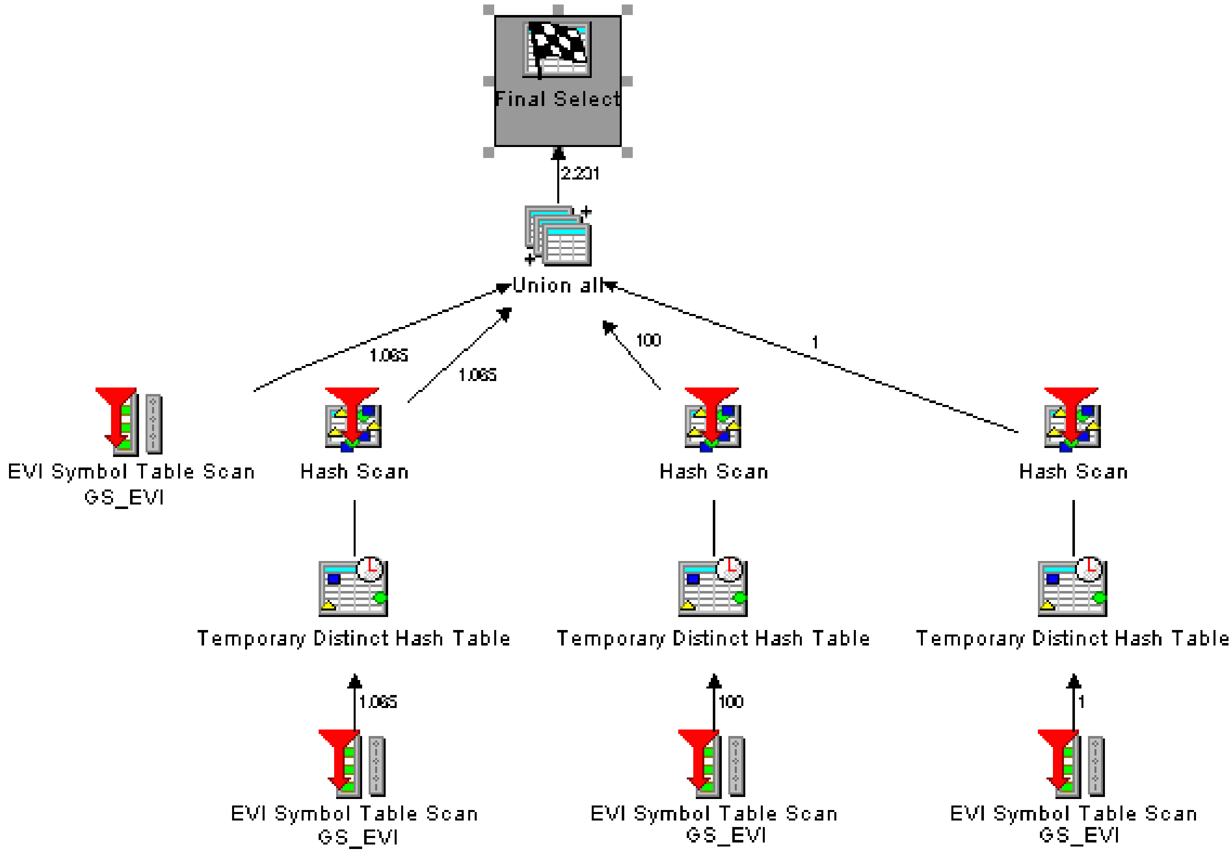
For example on the ROLLUP query below, the grouping is the sum of quantity rolled up at various levels (month, quarter, year) and ONLY the symbol table of the encoded vector index is accessed in the access plan.

```
SELECT year(shipdate) year_ship, quarter(shipdate) quarter, month(shipdate) month_ship,
sum(quantity)
 as totquantity
FROM  item_fact
GROUP BY ROLLUP (Year(shipdate), Quarter(shipdate), month(shipdate));
```

Here is the EVI INCLUDE create that will facilitate the rollup query.

```
CREATE ENCODED VECTOR INDEX GS_EVI
    ON   ITEM_FACT
    ( YEAR ( SHIPDATE ) ASC , QUARTER ( SHIPDATE ) ASC , MONTH ( SHIPDATE ) ASC )
    INCLUDE (  SUM ( QUANTITY ) , COUNT ( * ) );
```

The following graphic shows a Visual Explain that illustrates the access and the performance. Instead of accessing potentially millions of rows, the access is over a rather modest size symbol table.



Consider SMP and parallel index creation and maintenance

Symmetrical Multiprocessing (SMP) is a valuable tool for building and maintaining indexes in parallel. The results of using the optional SMP feature of IBM i are faster index build times, and faster I/O velocities while maintaining indexes in parallel. Using an SMP degree value of either *OPTIMIZE or *MAX, additional multiple tasks and additional system resources are used to build or maintain the indexes. With a degree value of *MAX, expect linear scalability on index creation. For example, creating indexes on a 4-processor system can be four times as fast as a 1-processor system.

Checking values in the overflow area

You can also use the **`Display File Description (DSPFD)`** command (or System i Navigator - Database) to check how many values are in the overflow area. Once the **DSPFD** command is issued, check the overflow area parameter for details on the initial and actual number of distinct key values in the overflow area.

Using CHGLF to rebuild the access path of an index

Use the **`Change Logical File (CHGLF)`** command with the attribute Force Rebuild Access Path set to *YES* (FRCRBDAP(*YES)). This command accomplishes the same thing as dropping and recreating the index, but it does not require that you know about how the index was built. This command is especially effective for applications where the original index definitions are not available, or for refreshing the access path.

Related information

SQL Create Index statement

SQL INCLUDE statement

Change Logical File (CHGLF) command

Display File Description (DSPFD) command

Comparing binary radix indexes and encoded vector indexes

DB2 for IBM i makes indexes a powerful tool.

The following table summarizes some of the differences between binary radix indexes and encoded vector indexes:

Table 51. Comparison of radix and EVI indexes

Comparison value	Binary Radix Indexes	Encoded Vector Indexes
Basic data structure	A wide, flat tree	A Symbol Table and a vector
Interface for creating	Command, SQL, System i Navigator	SQL, System i Navigator
Can be created in parallel	Yes	Yes
Can be maintained in parallel	Yes	Yes
Used for statistics	Yes	Yes
Used for selection	Yes	Yes, with dynamic bitmaps or RRN list
Used for joining	Yes	Yes (with a hash table)
Used for grouping	Yes	Yes
Used for ordering	Yes	No
Used to enforce unique Referential Integrity constraints	Yes	No
Source for predetermined or ready-made numeric aggregate results	No	Yes, with INCLUDE keyword option on create

Indexes & the optimizer

Since the optimizer uses cost based optimization, more information about the database rows and columns makes for a more efficient access plan created for the query. With the information from the indexes, the optimizer can make better choices about how to process the request (local selection, joins, grouping, and ordering).

The CQE optimizer attempts to examine most, if not all, indexes built over a table unless or until it times out. However, the SQE optimizer only considers those indexes that are returned by the Statistics Manager. These include only indexes that the Statistics Manager decides are useful in performing local selection based on the "where" clause predicates. Consequently, the SQE optimizer does not time out.

The primary goal of the optimizer is to choose an implementation that efficiently eliminates the rows that are not interesting or required to satisfy the request. Normally, query optimization is thought of as trying to find the rows of interest. A proper indexing strategy assists the optimizer and database engine with this task.

Instances where an index is not used

Db2 for i does not use indexes in the certain instances.

- For a column that is expected to be updated; for example, when using SQL, your program might include the following:

```
EXEC SQL
 DECLARE DEPTEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
      WHERE (WORKDEPT = 'D11' OR
               WORKDEPT = 'D21') AND
              EMPNO = '000190'
```

```
      FOR UPDATE OF EMPNO, WORKDEPT
END-EXEC.
```

When using the **OPNQRYF** command, for example:

```
OPNQRYF FILE((CORPDATA/EMPLOYEE)) OPTION(*ALL)
   QRYSLT('(WORKDEPT *EQ ''D11'' *OR WORKDEPT *EQ ''D21'')
   *AND EMPNO *EQ ''000190''')
```

Even if you do not intend to update the employee department, the system cannot use an index with a key of WORKDEPT.

An index can be used if all the index updatable columns are also used within the query as an isolatable selection predicate with an equal operator. In the previous example, the system uses an index with a key of EMPNO.

The system can operate more efficiently if the FOR UPDATE OF column list only names the column you intend to update: *WORKDEPT*. Therefore, do not specify a column in the FOR UPDATE OF column list unless you intend to update the column.

If you have an updatable cursor because of dynamic SQL, or FOR UPDATE was not specified and the program contains an UPDATE statement, then all columns can be updated.

- For a column being compared with another column from the same row. For example, when using SQL, your program might include the following:

```
EXEC SQL
 DECLARE DEPTDATA CURSOR FOR
    SELECT WORKDEPT, DEPTNAME
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = ADMRDEPT
END-EXEC.
```

When using the OPNQRYF command, for example:

```
OPNQRYF FILE (EMPLOYEE) FORMAT(FORMAT1)
   QRYSLT('WORKDEPT *EQ ADMRDEPT')
```

Even though there is an index for *WORKDEPT* and another index for *ADMRDEPT*, Db2 for i does not use either index. The index has no added benefit because every row of the table needs to be looked at.

Display indexes for a table
You can display indexes that are created on a table using System i Navigator.

To display indexes for a table, follow these steps:

1. In the System i Navigator window, expand the system that you want to use.
2. Expand **Databases** and the database that you want to work with.
3. Expand **Schemas** and the schema that you want to work with.
4. Right-click a table and select **Show Indexes**.

The Show index window includes the following columns:

Table 52. Columns used in Show index window

Column name	Description
Name	The SQL name for the index

Column name	Description
Type	The type of index displayed. Possible values are: • Keyed Physical File • Keyed Logical File • Primary Key Constraint • Unique Key Constraint • Foreign Key Constraint • Index
Schema	Schema or library containing the index or access path
Owner	User ID of the owner of this index or access path
System Name	System table name for the index or access path.
Text	The text description of the index or access path
Index partition	Partition detail for the index. Possible values: • <blank>, For all partitions • For Each Partition • specific name of the partition
Valid	Whether the access path or index is valid. The possible values are Yes or No.
Creation Date	The timestamp of when the index was created.
Last Build	The last time that the access path or index was rebuilt.
Last Query Use	Timestamp when the access path was last used by the optimizer.
Last Query Statistics Use	Timestamp when the access path was last used for statistics
Query Use Count	Number of times the access path has been used for a query
Query Statistics Use Count	Number of times the access path has been used for statistics
Last Used Date	Timestamp when the access path or index was last used.
Days Used Count	The number of days the index has been used.
Date Reset Days Used Count	The year and date when the days-used count was last set to 0.
Number of Key Columns	The number of key columns defined for the access path or index.
Key Columns	The key columns defined for the access path or index.
Current Key Values	The number of current key values.
Current Size	The size of the access path or index.
Current Allocated Pages	The current number of pages allocated for the access path or index.

Table 52. Columns used in Show index window (continued)

Table 52. Columns used in Show index window (continued)	
Column name	**Description**
Logical Page Size	The number of bytes used for the access path or the logical page size of the index. Indexes with larger logical page sizes are typically more efficient when scanned during query processing. Indexes with smaller logical page sizes are typically more efficient for simple index probes and individual key look ups. If the access path or index is an encoded vector, the value 0 is returned.
Duplicate Key Order	How the access path or index handles duplicate key values. Possible values are: • Unique - all values are unique. • Unique where not null - all values are unique unless null is specified.
Maximum Key Length	The maximum key length for the access path or index.
Unique Partial Key Values	The number of unique partial keys for the key fields 1 through 4. If the access path is an encoded vector, this number represents the number of full key distinct values.
Overflow Values	The number of overflow values for this encoded vector index.
Key Code Size	The length of the code assigned to each distinct key value of the encoded vector index.
Sparse	Is the index considered sparse. Sparse indexes only contain keys for rows that satisfy the query. Possible values are: • Yes • No
Derived Key	Is the index considered derived. A derived key is a key that is the result of an operation on the base column. Possible values are: • Yes • No
Partitioned	Is the index partition created for each data partition defined for the table using the specified columns. Possible values are: • Yes • No
Maximum Size	The maximum size of the access path or index.
Sort Sequence	The alternate character sorting sequence for National Language Support (NLS).
Language Identifier	The language code for the object.
Estimated Rebuild Time	The estimated time in seconds required to rebuild the access path or index.

Table 52. Columns used in Show index window (continued)	
Column name	**Description**
Held	Is a rebuild of an access path or index held. Possible values are: • Yes • No
Maintenance	For objects with key fields or join logical files, the type of access path maintenance used. The possible values are: • Do not wait • Delayed • Rebuild
Delayed Maintenance Keys	The number of delayed maintenance keys for the access path or index.
Recovery	When the access path is rebuilt after damage to the access path is recognized. The possible values are: • After IPL • During IPL • Next Open
Index Logical Reads	The number of access path or index logical read operations since the last IPL.
WHERE Clause	Specifies the condition to apply for a row to be included in the index.
WHERE Clause Has UDF	Does the WHERE clause have a UDF. Possible values are: • Yes • No
Table	Table name of the table that the index is based on.
Table Partition	Partition name of the table that the index is based on.
Table System Name	System name of the table that the index is based on.
Last Rebuild Number Keys	Number of keys in the index when the index was last rebuilt.
Last Rebuild Parallel Degree	Parallel degree used when the index was last rebuilt.
Last Rebuild Time	Amount of time in seconds it took to rebuild the index the last time the index was rebuilt.
Keep in Memory	Is the index kept in memory. Possible values are: • Yes • No
Sort Sequence Schema	Schema of the sort sequence table if one is used.
Sort Sequence Name	Name of the sort sequence table if one is used.
Random Reads	The number of reads that have occurred in a random fashion. Random means that the location of the row or key could not be predicted ahead of time.

Table 52. Columns used in Show index window (continued)	
Column name	**Description**
Media Preference	Indicates preference whether the storage for the table, partition, or index is allocated on Solid State Disk (SSD), if available.

Determine unnecessary indexes
You can easily determine which indexes are being used for query optimization.

Before V5R3, it was difficult to determine unnecessary indexes. Using the Last Used Date was not dependable, as it was only updated when the logical file was opened using a native database application (for example, an RPG application). Furthermore, it was difficult to find all the indexes over a physical file. Indexes are created as part of a keyed physical file, keyed logical file, join logical file, SQL index, primary key or unique constraint, or referential constraint. However, you can now easily find all indexes and retrieve statistics on index usage as a result of System i Navigator and IBM i functionality. To assist you in tuning your performance, this function now produces statistics on index usage as well as index usage in a query.

To access index information through the System i Navigator, navigate to: **Database** > **Schemas** > **Tables**. Right-click your table and select **Show Indexes**.

You can show all indexes for a schema by right-clicking on **Tables** or **Indexes** and selecting Show indexes.

Note: You can also view the statistics through the Retrieve Member Description (QUSRMBRD) API.

Certain fields available in the **Show Indexes** window can help you to determine any unnecessary indexes. Those fields are:

Last Query Use
States the timestamp when the index was last used to retrieve data for a query.

Last Query Statistic Use
States the timestamp when the index was last used to provide statistical information.

Query Use Count
Lists the number of instances the index was used in a query.

Query Statistics Use
Lists the number of instances the index was used for statistical information.

Last Used Date
The century and date this index was last used.

Days Used Count
The number of days the index was used. If the index does not have a last used date, the count is 0.

Date Reset Days Used Count
The date that the days used count was last reset. You can reset the days used by **`Change Object Description (CHGOBJD)`** command.

The fields start and stop counting based on your situation, or the actions you are currently performing on your system. The following list describes what might affect one or both of your counters:

- The SQE and CQE query engines increment both counters. As a result, the statistics field is updated regardless of which query interface is used.
- A save and restore procedure does not reset the statistics counter if the index is restored over an existing index. If an index is restored that does not exist on the system, the statistics are reset.

Related information
Retrieve Member Description (QUSRMBRD) API
Change Object Description (CHGOBJD) command

Reset usage counts
Resetting the usage counts for a table allows you to determine how the changes you made to your indexing strategy affected the indexes and constraints on that table. For example, if your new strategy causes an index to never be used, you could then delete that index. Resetting usage counts on a table affect all indexes and constraints that are created on that object.

Note: Resetting usage counts for a keyed physical file or a constraint in the Show Indexes window resets the counts of all constraints and keyed access for that file or table.

You can reset index usage counts by right-clicking a specific index in the Indexes folder or in the Show Indexes dialog and selecting **Reset Usage Counts**.

View index build status
You can view a list of indexes that are being built by the database. This view might be helpful in determining when the index becomes usable to your applications.

To display indexes that are being built, follow these steps:

1. In the System i Navigator window, expand the system that you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with and then expand the Database Maintenance folder. Select **Index Builds**.

Manage index rebuilds
You can manage the rebuild of your indexes using System i Navigator. You can view a list of access paths that are rebuilding and either hold the access path rebuild or change the priority of a rebuild.

To display access paths to rebuild, follow these steps:

1. In the System i Navigator window, expand the system that you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with and then expand the **Database Maintenance** folder. Select **Index Rebuilds**.

The access paths to rebuild dialog includes the following columns:

Table 53. Columns used in Index rebuilds window

Column name	Description
Name	Name of access path being rebuilt.
Schema	Schema name where the index is located.
System Name	The system name of the file that owns the index to be rebuilt.
System Schema	System schema name of access path being rebuilt.
Type	The type of index displayed. Possible values are: Keyed Physical File Keyed Logical File Primary Key Unique Key Foreign Key Index

Table 53. Columns used in Index rebuilds window (continued)

Column name	Description
Status	Displays the status of the rebuild. Possible values are: 1-99 – *Rebuild Priority* Running – *Rebuilding* Held – *Held from be rebuilt*
Rebuild Priority	Displays the priority in which the rebuild for this access path is run. Also referred to as sequence number. Possible values are: 1-99: Order to rebuild Held Open
Rebuild Reason	Displays the reason why this access path needs to be rebuilt. Possible values are: Create or build index IPL Runtime error Change file or index sharing Other Not needed Change End of Data Restore Alter table Change table Change file Reorganize Enable a constraint Alter table recovery Change file recovery Index shared Runtime error Verify constraint Convert member Restore recovery

Table 53. Columns used in Index rebuilds window (continued)	
Column name	**Description**
Rebuild Reason Subtype	Displays the subtype reason why this access path needs to be rebuilt. Possible values are: Unexpected error Index in use during failure Unexpected error during update, delete, or insert Delayed maintenance overflow or catch-up error Other No event Change End of Data Delayed maintenance mismatch Logical page size mismatch Partial index restore Index conversion Index not saved and restored Partitioning mismatch Partitioning change Index or key attributes change Original index invalid Index attributes change Force rebuild of index Index not restored Asynchronous rebuilds requested Job ended abnormally Alter table Change constraint Index invalid or attributes change Invalid unique index found Invalid constraint index found Index conversion required If there is no subtype, this field displays 0.

Table 53. Columns used in Index rebuilds window (continued)	
Column name	**Description**
Invalidation Reason	Displays the reason why this access path was invalidated. Possible values are: User requested (See Invalidation Reason type for more information) Create or build Index Load (See Invalidation Reason type for more information) Initial Program Load (IPL) Runtime error Modify Journal failed to build the index Marked index as fixable during runtime Marked index as fixable during IPL Change end of data

Table 53. Columns used in Index rebuilds window (continued)	
Column name	**Description**
Invalidation Reason Type	Displays the reason type for why this access path was invalidation. Possible reason types for User requested: Invalid because of REORG It is a copy Alter file Converting new member Change to *FRCRBDAP Change to *UNIQUE Change to *REBLD Possible reason type for LOAD The index was marked for invalidation but the system crashed before the invalidation could actually occur The index was associated with the overlaid data space header during a load, therefore it was invalidated Index was in IMPI format. The header was converted and now it is invalidated to be rebuilt in RISC format The RISC index was converted to V5R1 format Index invalidated due to partial load Index invalidated due to a delayed maintenance mismatch Index invalidated due to a pad key mismatch Index invalidated due to a significant fields bitmap fix Index invalidated due to a logical page size mismatch Index was not restored. File might have been saved with ACCPTH(*NO) or index did not exist when file was saved. Index was not restored. File might have been saved with ACCPTH(*NO) or index did not exist when file was saved. Index was rebuilt because file was saved in an inconsistent state with SAVACT(*SYSDFN). For other invalidation codes, this field displays 0.
Estimated Rebuild Time	Estimated amount of time in seconds that it takes to rebuild the index access path.
Rebuild Start Time	Time when the rebuild was started.

Table 53. Columns used in Index rebuilds window (continued)	
Column name	**Description**
Elapsed Rebuild Time	Amount of time that has elapsed in seconds since the start of the rebuild of the access path.
Unique	Indicates whether the rows in the access path are unique. Possible values are: Yes No
Last Query Use	Timestamp when the access path was last used
Last Query Statistics Use	Timestamp when the access path was last used for statistics
Query Use Count	Number of times the access path has been used for a query
Query Statistics Use Count	Number of times the access path has been used for statistics
Partition	Partition detail for the index. Possible values: • <blank>, which means For all partitions • For Each Partition • specific name of the partition
Owner	User ID of the owner of this access path.
Parallel Degree	Number of processors to be used to rebuild the index.
Text	Text description of the file owning the index.

You can also use the **`Edit Rebuild of Access Paths (EDTRBDAP)`** command to manage rebuilding of access paths.

Related information

Rebuild access paths

Edit Rebuild of Access Paths (EDTRBDAP) command

Indexing strategy

There are two approaches to index creation: proactive and reactive. Proactive index creation involves anticipating which columns are most often used for selection, joining, grouping, and ordering. Then building indexes over those columns. In the reactive approach, indexes are created based on optimizer feedback, query implementation plan, and system performance measurements.

It is useful to initially build indexes based on the database model and applications and not any particular query. As a starting point, consider designing basic indexes based on the following criteria:

- Primary and foreign key columns based on the database model
- Commonly used local selection columns, including columns that are dependent, such as an automobile's make and model
- Commonly used join columns not considered primary or foreign key columns
- Commonly used grouping columns

Related information

Indexing and statistics strategies for DB2 for i5/OS

Reactive approach to tuning

To perform reactive tuning, build a prototype of the proposed application without any indexes and start running some queries. Or you could build an initial set of indexes and start running the application to see which ones get used and which do not. Even with a smaller database, the slow running queries become obvious quickly.

The reactive tuning method is also used when trying to understand and tune an existing application that is not performing up to expectations. Use the appropriate debugging and monitoring tools, described in the next section, to view the database feedback messages:

- the indexes the optimizer recommends for local selection
- the temporary indexes used for a query
- the query implementation methods the optimizer chose

If the database engine is building temporary indexes to process joins or perform grouping and selection over permanent tables, build permanent indexes over the same columns. This technique is used to eliminate the temporary index creation. In some cases, a temporary index is built over a temporary table, so a permanent index is not able to be built for those tables. You can use the optimization tools listed in the previous section to note the temporary index creation, the reason it was created, and the key columns.

Proactive approach to tuning

Typically you will create an index for the most selective columns and create statistics for the least selective columns in a query. By creating an index, the optimizer knows that the column is selective and it also gives the optimizer the ability to use that index to implement the query.

In a perfect radix index, the order of the columns is important. In fact, it can make a difference as to whether the optimizer uses it for data retrieval at all. As a general rule, order the columns in an index in the following way:

- Equal predicates first. That is, any predicate that uses the "=" operator may narrow down the range of rows the fastest and should therefore be first in the index.
- If all predicates have an equal operator, then order the columns as follows:
  - Selection predicates + join predicates
  - Join predicates + selection predicates
  - Selection predicates + group by columns
  - Selection predicates + order by columns

In addition to the guidelines above, in general, the most selective key columns should be placed first in the index.

Consider the following SQL statement:

```
SELECT b.col1, b.col2, a.col1
   FROM table1 a, table2 b
   WHERE b.col1='some_value' AND
     b.col2=some_number AND
     a.join_col=b.join_col
   GROUP BY b.col1, b.col2, a.col1
   ORDER BY b.col1
```

With a query like this, the proactive index creation process can begin. The basic rules are:

- Custom-build a radix index for the largest or most commonly used queries. Example using the query above:

  ```
  radix index over join column(s) - a.join_col and b.join_col
  radix index over most commonly used local     selection column(s) - b.col2
  ```

- For ad hoc online analytical processing (OLAP) environments or less frequently used queries, build single-key EVIs over the local selection column(s) used in the queries. Example using the query above:

  ```
  EVI over non-unique local selection columns - b.col1 and b.col2
  ```

Coding for effective indexes

The following topics provide suggestions to help you design code which allows Db2 for i to take advantage of available indexes:

Avoid numeric conversions

When a column value and a host variable (or constant value) are being compared, try to specify the same data types and attributes. Db2 for i might not use an index for the named column if the host variable or constant value has a greater precision than the precision of the column. If the two items being compared have different data types, Db2 for i needs to convert one or the other of the values, which can result in inaccuracies (because of limited machine precision).

To avoid problems for columns and constants being compared, use the following:

- same data type
- same scale, if applicable
- same precision, if applicable

For example, EDUCLVL is a halfword integer value (SMALLINT). When using SQL, specify:

```
… WHERE EDUCLVL < 11 AND
                  EDUCLVL >= 2
```

instead of

```
… WHERE EDUCLVL < 1.1E1 AND
                  EDUCLVL > 1.3
```

When using the OPNQRYF command, specify:

```
... QRYSLT('EDUCLVL *LT 11 *AND ENUCLVL *GE 2')
```

instead of

```
... QRYSLT('EDUCLVL *LT 1.1E1 *AND EDUCLVL *GT 1.3')
```

If an index was created over the EDUCLVL column, then the optimizer might not use the index in the second example. The constant precision is greater than the column precision. It attempts to convert the constant to the precision of the column. In the first example, the optimizer considers using the index, because the precisions are equal.

Avoid arithmetic expressions

Do not use an arithmetic expression as an operand to compare to a column in a row selection predicate. The optimizer does not use an index on a column compared to an arithmetic expression. While this technique might not cause the column index to become unusable, it prevents any estimates and possibly the use of index scan-key positioning. The primary thing that is lost is the ability to use and extract any statistics that might be useful in the optimization of the query.

For example, when using SQL, specify the following:

```
… WHERE SALARY > 16500
```

instead of

```
… WHERE SALARY > 15000*1.1
```

Avoid character string padding
Try to use the same data length when comparing a fixed-length character string column value to a host variable or constant value. Db2 for i might not use an index if the constant value or host variable is longer than the column length.

For example, EMPNO is CHAR(6) and DEPTNO is CHAR(3). For example, when using SQL, specify the following:

```
… WHERE EMPNO > '000300' AND
                 DEPTNO < 'E20'
```

instead of

```
… WHERE EMPNO > '000300 ' AND
                 DEPTNO < 'E20 '
```

When using the OPNQRYF command, specify:

```
... QRYSLT('EMPNO *GT "000300" *AND DEPTNO *LT "E20"')
```

instead of

```
... QRYSLT('EMPNO *GT "000300" *AND DEPTNO *LT "E20"')
```

Avoid the use of LIKE patterns beginning with % or _
The percent (%), and underline (_), used in the pattern of a LIKE (OPNQRYF %WLDCRD) predicate, specify a character string like the row column values to select. They can take advantage of indexes when used to denote characters in the middle or at the end of a character string.

For example, when using SQL, specify the following:

```
… WHERE LASTNAME LIKE 'J%SON%'
```

When using the OPNQRYF command, specify the following:

```
... QRYSLT('LASTNAME *EQ %WLDCRD(''J*SON*'')')
```

However, when used at the beginning of a character string, they can prevent Db2 for i from using any indexes that might be defined on the LASTNAME column to limit the number of rows scanned using index scan-key positioning. Index scan-key selection, however, is allowed. For example, in the following queries index scan-key selection can be used, but index scan-key positioning cannot.

In SQL:

```
… WHERE LASTNAME LIKE '%SON'
```

In OPNQRYF:

```
… QRYSLT('LASTNAME *EQ %WLDCRD(''*SON'')')
```

Avoid patterns with a % so that you can get the best performance with key processing on the predicate. If possible, try to get a partial string to search so that index scan-key positioning can be used.

For example, if you were looking for the name "Smithers", but you only type "S%," this query returns all names starting with "S." Adjust the query to return all names with "Smi%". By forcing the use of partial strings, you might get better performance in the long term.

Using derived indexes

SQL indexes can be created where the key is specified as an expression. This type of key is also referred to as a derived key.

For example, look at the following:

```
CREATE INDEX TOTALIX ON EMPLOYEE(SALARY+BONUS+COMM AS TOTAL)
```

In this example, return all the employees whose total compensation is greater than 50000.

```
SELECT * FROM EMPLOYEE
 WHERE SALARY+BONUS+COMM > 50000
 ORDER BY SALARY+BONUS+COMM
```

Since the optimizer uses the index TOTALIX with index probe to satisfy the WHERE selection and the ordering criteria.

Some special considerations to with derived key index usage and matching include:

- There is no matching for index key constants to query host variables. This non-match includes implicit parameter marker conversion performed by the database manager.

  ```
  CREATE INDEX D_IDX1 ON EMPLOYEE (SALARY/12 AS MONTHLY)
  ```

  In this example, return all employees whose monthly salary is greater than 3000.

  long months = 12;

  ```
  EXEC SQL SELECT * FROM EMPLOYEE WHERE SALARY/:months > 3000
  ```

  However, in this case the optimizer does not use the index since there is no support for matching the host variable value months in the query to the constant 12 in the index.

  Usage of the QAQQINI option PARAMETER_MARKER_CONVERSION with value *NO can be used to prevent conversion of constants to parameter markers. This technique allows for improved derived index key matching. However, because of the performance implications of using this QAQQINI setting, take care with its usage.

- In general, expressions in the index must match the expression in the query:

  ```
  .... WHERE SALARY+COMM+BONUS > 50000
  ```

  In this case, the WHERE SALARY+COMM+BONUS is different from the index key SALARY+BONUS +COMM and would not match.

- It is recommended that the derived index keys be kept as simple as possible. The more complex the query expression to match and the index key expression is, the less likely it is that the index is used.
- The CQE optimizer has limited support for matching derived key indexes.

Related reference

Derived key index
You can use the SQL CREATE INDEX statement to create a derived key index using an SQL expression.

Related information

SQL Create Index statement

Using sparse indexes

SQL indexes can be created using WHERE selection predicates. These indexes can also be referred to as sparse indexes. The advantage of a sparse index is that fewer entries are maintained in the index. Only those entries matching the WHERE selection criteria are maintained in the index.

In general, the query WHERE selection must be a subset of the sparse index WHERE selection in order for the sparse index to be used.

Here is a simple example of when a sparse index can be used:

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

It is recommended that the WHERE selection in the sparse index is kept as simple as possible. The more complex the WHERE selection, the more difficult it becomes to match the sparse index WHERE selection to the query WHERE selection. Then it is less likely that the sparse index is used. The CQE optimizer does not support sparse indexes. It does support select/omit logical files however. The SQE optimizer matches the CQE optimizer in its support for select/omit logical files and has nearly full support for sparse indexes.

Related reference

Sparse indexes
You can use the SQL CREATE INDEX statement to create a sparse index using SQL selection predicates.

Related information

SQL Create Index statement

Using indexes with sort sequence

The following sections provide useful information about how indexes work with sort sequence tables.

Using indexes and sort sequence with selection, joins, or grouping
Before using an existing index, Db2 for i ensures the attributes of the columns (selection, join, or grouping columns) match the attributes of the key columns in the existing index. The sort sequence table is an additional attribute that must be compared.

The query sort sequence table (specified by the SRTSEQ and LANGID) must match the index sort sequence table. Db2 for i compares the sort sequence tables. If they do not match, the existing index cannot be used.

There is an exception to this rule, however. If the sort sequence table associated with the query is a unique-weight sequence table (including *HEX), Db2 for i acts as though no sort sequence table is specified for selection, join, or grouping columns that use the following operators and predicates:

- equal (=) operator
- not equal (^= or <>) operator
- LIKE predicate (OPNQRYF %WLDCRD and *CT)
- IN predicate (OPNQRYF %VALUES)

When these conditions are true, Db2 for i is free to use any existing index where the key columns match the columns and either:

- The index does not contain a sort sequence table or
- The index contains a unique-weight sort sequence table

Note:

1. The table does not need to match the unique-weight sort sequence table associated with the query.
2. Bitmap processing has a special consideration when multiple indexes are used for a table. If two or more indexes have a common key column referenced in the query selection, then those indexes must either use the same sort sequence table or no sort sequence table.

Using indexes and sort sequence with ordering
Unless the optimizer chooses a sort to satisfy the ordering request, the index sort sequence table must match the query sort sequence table.

When a sort is used, the translation is done during the sort. Since the sort is handling the sort sequence requirement, this technique allows Db2 for i to use any existing index that meets the selection criteria.

Index examples

The following index examples are provided to help you create effective indexes.

For the purposes of the examples, assume that three indexes are created.

Assume that an index HEXIX was created with *HEX as the sort sequence.

```
CREATE INDEX HEXIX ON STAFF (JOB)
```

Assume that an index UNQIX was created with a unique-weight sort sequence.

```
CREATE INDEX UNQIX ON STAFF (JOB)
```

Assume that an index SHRIX was created with a shared-weight sort sequence.

```
CREATE INDEX SHRIX ON STAFF (JOB)
```

Index example: Equal selection with no sort sequence table

Equal selection with no sort sequence table (SRTSEQ(*HEX)).

```
SELECT * FROM STAFF
  WHERE JOB = 'MGR'
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   SRTSEQ(*HEX)
```

The system can use either index HEXIX or index UNQIX.

Index example: Equal selection with a unique-weight sort sequence table

Equal selection with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
SELECT * FROM STAFF
 WHERE JOB = 'MGR'
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

The system can use either index HEXIX or index UNQIX.

Index example: Equal selection with a shared-weight sort sequence table

Equal selection with a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

```
SELECT * FROM STAFF
 WHERE JOB = 'MGR'
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

The system can only use index SHRIX.

Index example: Greater than selection with a unique-weight sort sequence table

Greater than selection with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
SELECT * FROM STAFF
 WHERE JOB > 'MGR'
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *GT ''MGR''')
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

The system can only use index UNQIX.

Index example: Join selection with a unique-weight sort sequence table
Join selection with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
  SELECT * FROM STAFF S1, STAFF S2
   WHERE S1.JOB = S2.JOB
```

or the same query using the JOIN syntax.

```
  SELECT *
   FROM STAFF S1 INNER JOIN STAFF S2
         ON S1.JOB = S2.JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE(STAFF STAFF)
   FORMAT(FORMAT1)
   JFLD((1/JOB 2/JOB *EQ))
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

The system can use either index HEXIX or index UNQIX for either query.

Index example: Join selection with a shared-weight sort sequence table
Join selection with a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

```
  SELECT * FROM STAFF S1, STAFF S2
   WHERE S1.JOB = S2.JOB
```

or the same query using the JOIN syntax.

```
  SELECT *
   FROM STAFF S1 INNER JOIN STAFF S2
         ON S1.JOB = S2.JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE(STAFF STAFF) FORMAT(FORMAT1)
   JFLD((1/JOB 2/JOB *EQ))
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

The system can only use index SHRIX for either query.

Index example: Ordering with no sort sequence table
Ordering with no sort sequence table (SRTSEQ(*HEX)).

```
  SELECT * FROM STAFF
   WHERE JOB = 'MGR'
   ORDER BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   KEYFLD(JOB)
   SRTSEQ(*HEX)
```

The system can only use index HEXIX.

Index example: Ordering with a unique-weight sort sequence table
Ordering with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
   SELECT * FROM STAFF
    WHERE JOB = 'MGR'
    ORDER BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   KEYFLD(JOB) SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

The system can only use index UNQIX.

Index example: Ordering with a shared-weight sort sequence table
Ordering with a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

```
   SELECT * FROM STAFF
    WHERE JOB = 'MGR'
    ORDER BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   KEYFLD(JOB) SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

The system can only use index SHRIX.

Index example: Ordering with ALWCPYDTA(*OPTIMIZE) and a unique-weight sort sequence table
Ordering with ALWCPYDTA(*OPTIMIZE) and a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
   SELECT * FROM STAFF
    WHERE JOB = 'MGR'
    ORDER BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF))
   QRYSLT('JOB *EQ ''MGR''')
   KEYFLD(JOB)
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
   ALWCPYDTA(*OPTIMIZE)
```

The system can use either index HEXIX or index UNQIX for selection. Ordering is done during the sort using the *LANGIDUNQ sort sequence table.

Index example: Grouping with no sort sequence table
Grouping with no sort sequence table (SRTSEQ(*HEX)).

```
   SELECT JOB FROM STAFF
    GROUP BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT2)
   GRPFLD((JOB))
   SRTSEQ(*HEX)
```

The system can use either index HEXIX or index UNQIX.

Index example: Grouping with a unique-weight sort sequence table

Grouping with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
SELECT JOB FROM STAFF
 GROUP BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT2)
   GRPFLD((JOB))
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

The system can use either index HEXIX or index UNQIX.

Index example: Grouping with a shared-weight sort sequence table

Grouping with a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

```
SELECT JOB FROM STAFF
 GROUP BY JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT2)
   GRPFLD((JOB))
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

The system can only use index SHRIX.

The following examples assume that three more indexes are created over columns JOB and SALARY. The CREATE INDEX statements precede the examples.

Assume an index HEXIX2 was created with *HEX as the sort sequence.

```
CREATE INDEX HEXIX2 ON STAFF (JOB, SALARY)
```

Assume that an index UNQIX2 was created and the sort sequence is a unique-weight sort sequence.

```
CREATE INDEX UNQIX2 ON STAFF (JOB, SALARY)
```

Assume an index SHRIX2 was created with a shared-weight sort sequence.

```
CREATE INDEX SHRIX2 ON STAFF (JOB, SALARY)
```

Index example: Ordering and grouping on the same columns with a unique-weight sort sequence table

Ordering and grouping on the same columns with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
SELECT JOB, SALARY FROM STAFF
 GROUP BY JOB, SALARY
 ORDER BY JOB, SALARY
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(JOB SALARY)
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
```

The system can use UNQIX2 to satisfy both the grouping and ordering requirements. If index UNQIX2 did not exist, the system creates an index using a sort sequence table of *LANGIDUNQ.

Index example: Ordering and grouping on the same columns with ALWCPYDTA(*OPTIMIZE) and a unique-weight sort sequence table

Ordering and grouping on the same columns with ALWCPYDTA(*OPTIMIZE) and a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
   SELECT JOB, SALARY FROM STAFF
    GROUP BY JOB, SALARY
    ORDER BY JOB, SALARY
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(JOB SALARY)
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
   ALWCPYDTA(*OPTIMIZE)
```

The system can use UNQIX2 to satisfy both the grouping and ordering requirements. If index UNQIX2 did not exist, the system does one of the following actions:

- Create an index using a sort sequence table of *LANGIDUNQ or
- Use index HEXIX2 to satisfy the grouping and to perform a sort to satisfy the ordering

Index example: Ordering and grouping on the same columns with a shared-weight sort sequence table

Ordering and grouping on the same columns with a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

```
   SELECT JOB, SALARY FROM STAFF
    GROUP BY JOB, SALARY
    ORDER BY JOB, SALARY
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(JOB SALARY)
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

The system can use SHRIX2 to satisfy both the grouping and ordering requirements. If index SHRIX2 did not exist, the system creates an index using a sort sequence table of *LANGIDSHR.

Index example: Ordering and grouping on the same columns with ALWCPYDTA(*OPTIMIZE) and a shared-weight sort sequence table

Ordering and grouping on the same columns with ALWCPYDTA(*OPTIMIZE) and a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU).

```
   SELECT JOB, SALARY FROM STAFF
    GROUP BY JOB, SALARY
    ORDER BY JOB, SALARY
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(JOB SALARY)
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
   ALWCPYDTA(*OPTIMIZE)
```

The system can use SHRIX2 to satisfy both the grouping and ordering requirements. If index SHRIX2 did not exist, the system creates an index using a sort sequence table of *LANGIDSHR.

Index example: Ordering and grouping on different columns with a unique-weight sort sequence table

Ordering and grouping on different columns with a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
   SELECT JOB, SALARY FROM STAFF
    GROUP BY JOB, SALARY
    ORDER BY SALARY, JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(SALARY JOB)
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
```

The system can use index HEXIX2 or index UNQIX2 to satisfy the grouping requirements. A temporary result is created containing the grouping results. A temporary index is then built over the temporary result using a *LANGIDUNQ sort sequence table to satisfy the ordering requirements.

Index example: Ordering and grouping on different columns with ALWCPYDTA(*OPTIMIZE) and a unique-weight sort sequence table

Ordering and grouping on different columns with ALWCPYDTA(*OPTIMIZE) and a unique-weight sort sequence table (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

```
   SELECT JOB, SALARY FROM STAFF
    GROUP BY JOB, SALARY
    ORDER BY SALARY, JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(SALARY JOB)
   SRTSEQ(*LANGIDUNQ) LANGID(ENU)
   ALWCPYDTA(*OPTIMIZE)
```

The system can use index HEXIX2 or index UNQIX2 to satisfy the grouping requirements. A sort is performed to satisfy the ordering requirements.

Index example: Ordering and grouping on different columns with ALWCPYDTA(*OPTIMIZE) and a shared-weight sort sequence table

Ordering and grouping on different columns with ALWCPYDTA(*OPTIMIZE) and a shared-weight sort sequence table (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

```
   SELECT JOB, SALARY FROM STAFF
    GROUP BY JOB, SALARY
    ORDER BY SALARY, JOB
```

When using the **OPNQRYF** command, specify:

```
OPNQRYF FILE((STAFF)) FORMAT(FORMAT3)
   GRPFLD(JOB SALARY)
   KEYFLD(SALARY JOB)
   SRTSEQ(*LANGIDSHR) LANGID(ENU)
   ALWCPYDTA(*OPTIMIZE)
```

The system can use index SHRIX2 to satisfy the grouping requirements. A sort is performed to satisfy the ordering requirements.

Sparse index examples

This topic shows examples of how the sparse index matching algorithm works.

In example S1, the query selection is a subset of the sparse index selection and consequently an index scan over the sparse index is used. The remaining query selection (COL3=30) is executed following the index scan.

Example S1

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In example S2, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

Example S2

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20
```

In example S3, the query selection exactly matches the sparse index selection and an index scan over the sparse index can be used.

Example S3

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In example S4, the query selection is a subset of the sparse index selection and an index scan over the sparse index can be used. The remaining query selection (COL3=30) is executed following the index scan.

Example S4

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

In example S5, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

Example S5

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20
```

In example S6, the query selection exactly matches the sparse index selection and an index scan over the sparse index can be used. The query selection is executed following the index scan to eliminate excess records from the sparse index.

Example S6

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20 or COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20 or COL3=30
```

In example S7, the query selection is a subset of the sparse index selection and an index scan over the sparse index can be used. The query selection is executed following the index scan to eliminate excess records from the sparse index.

Example S7

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20 or COL3=30
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20
```

In example S8, the query selection is not a subset of the sparse index selection and the sparse index cannot be used.

Example S8

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 or COL2=20
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 or COL2=20 or COL3=30
```

In the next example S9, the constant 'MN' was replaced by a parameter marker for the query selection. The sparse index had the local selection of COL1='MN' applied to it when it was created. The sparse index matching algorithm matches the parameter marker to the constant 'MN' in the query predicate COL1 =?. It verifies that the value of the parameter marker is the same as the constant in the sparse index; therefore the sparse index can be used.

Example S9

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
Where Col3='WIN' and (Col1=? or Col2='TWINS')
```

In the next example S10, the keys of the sparse index match the order by fields in the query. For the sparse index to satisfy the specified ordering, the optimizer must verify that the query selection is a subset of the sparse index selection. In this example, the sparse index can be used.

Example S10

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL1, COL3)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
Where Col3='WIN' and (Col1='MN' or Col2='TWINS')
ORDER  BY COL1, COL3
```

In the next example S11, the keys of the sparse index do not match the order by fields in the query. But the selection in sparse index T2 is a superset of the query selection. Depending on size, the optimizer might choose an index scan over sparse index T2 and then use a sort to satisfy the specified ordering.

Example S11

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL2, COL4)
WHERE  COL1='MN' or COL2='TWINS'
```

```
SELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
Where Col3='WIN' and (Col1='MN' or Col2='TWINS')
ORDER  BY COL1, COL3
```

The next example S12 represents the classic optimizer decision: is it better to do an index probe using index IX1 or is it better to do an index scan using sparse index SPR1? Both indexes retrieve the same number of index entries and have the same cost from that point forward. For example, both indexes have the same cost to retrieve the selected records from the dataspace, based on the retrieved entries/keys.

The cost to retrieve the index entries is the deciding criteria. In general, if index IX1 is large then an index scan over sparse index SPR1 has a lower cost to retrieve the index entries. If index IX1 is rather small then an index probe over index IX1 has a lower cost to retrieve the index entries. Another cost decision is reusability. The plan using sparse index SPR1 is not as reusable as the plan using index IX1 because of the static selection built into the sparse selection.

Example S12

```
CREATE INDEX MYLIB/IX1 on MYLIB/T1 (COL1, COL2, COL3)
```

```
CREATE INDEX MYLIB/SPR1 on MYLIB/T1 (COL3)
WHERE  COL1=10 and COL2=20 and COL3=30
```

```
CSELECT COL1, COL2, COL3, COL4
FROM   MYLIB/T1
WHERE  COL1=10 and COL2=20 and COL3=30
```

Application design tips for database performance

There are some design tips that you can apply when designing SQL applications to maximize your database performance.

Using live data

The term *live data* refers to the type of access that the database manager uses when it retrieves data without making a copy of the data. Using this type of access, the data, which is returned to the program, always reflects the current values of the data in the database. The programmer can control whether the database manager uses a copy of the data or retrieves the data directly. This control is done by specifying the allow copy data (ALWCPYDTA) parameter on the precompiler commands or the **Start SQL (STRSQL)** command.

Specifying ALWCPYDTA(*NO) instructs the database manager to always use live data. In most cases, forcing live data access is a detriment to performance. It severely limits the possible plan choices that the optimizer could use to implement the query. Avoid it in most cases. However, in specialized cases involving a simple query, live data access can be used as a performance advantage. The cursor does not need to be closed and opened again to refresh the data being retrieved.

An example application demonstrating this advantage is one that produces a list on a display. If the display can show only 20 list elements at a time, then, after the initial 20 elements are displayed, the programmer can request that the next 20 rows be displayed. A typical SQL application designed for an operating system other than the IBM i operating system, might be structured as follows:

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
     ORDER BY EMPNO
END-EXEC.

EXEC SQL
    OPEN C1
END-EXEC.

*    PERFORM FETCH-C1-PARA  20 TIMES.

     MOVE EMPNO to LAST-EMPNO.

EXEC SQL
    CLOSE C1
END-EXEC.

*    Show the display and wait for the user to indicate that
*    the next 20 rows should be displayed.

EXEC SQL
    DECLARE C2 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
    WHERE EMPNO > :LAST-EMPNO
    ORDER BY EMPNO
END-EXEC.

EXEC SQL
    OPEN C2
END-EXEC.

*    PERFORM FETCH-C21-PARA  20 TIMES.

*    Show the display with these 20 rows of data.

EXEC SQL
    CLOSE C2
END-EXEC.
```

In the preceding example, notice that an additional cursor had to be opened to continue the list and to get current data. This technique can result in creating an additional ODP that increases the processing time on the system. In place of the preceding example, the programmer can design the application specifying ALWCPYDTA(*NO) with the following SQL statements:

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
     ORDER BY EMPNO
END-EXEC.

EXEC SQL
    OPEN C1
END-EXEC.

*    Display the screen with these 20 rows of data.

*    PERFORM FETCH-C1-PARA  20 TIMES.

*    Show the display and wait for the user to indicate that
*    the next 20 rows should be displayed.

*    PERFORM FETCH-C1-PARA  20 TIMES.

EXEC SQL
    CLOSE C1
END-EXEC.
```

In the preceding example, the query might perform better if the FOR 20 ROWS clause was used on the multiple-row FETCH statement. Then, the 20 rows are retrieved in one operation.

Related information

Start SQL Interactive Session (STRSQL) command

Reducing the number of open operations

The SQL data manipulation language statements must do database open operations in order to create an open data path (ODP) to the data. An open data path is the path through which all input/output operations for the table are performed. In a sense, it connects the SQL application to a table. The number of open operations in a program can significantly affect performance.

A database open operation occurs on:

- An OPEN statement
- SELECT INTO statement
- An INSERT statement with a VALUES clause
- An UPDATE statement with a WHERE condition
- An UPDATE statement with a WHERE CURRENT OF cursor and SET clauses that refer to operators or functions
- SET statement that contains an expression
- VALUES INTO statement that contains an expression
- A DELETE statement with a WHERE condition

An INSERT statement with a select-statement requires two open operations. Certain forms of subqueries could also require one open per subselect.

To minimize the number of opens, Db2 for i leaves the open data path (ODP) open and reuses the ODP if the statement is run again, unless:

- The ODP used a host variable to build a subset temporary index. The optimizer could choose to build a temporary index with entries for only the rows that match the row selection specified in the SQL statement. If a host variable was used in the row selection, the temporary index does not have the entries required for a different host variable value.
- Ordering was specified on a host variable value.
- An **Override Database File (OVRDBF)** or **Delete Override (DLTOVR)** CL command has been issued since the ODP was opened, which affects the SQL statement execution.

  Note: Only overrides that affect the name of the table being referred to causes the ODP to be closed within a given program invocation.
- The join is a complex join that requires temporary objects to contain the intermediate steps of the join.
- Some cases involve a complex sort, where a temporary file is required, might not be reusable.
- A change to the library list since the last open has occurred, which changes the table selected by an unqualified referral in system naming mode.
- The join was implemented by the CQE optimizer using hash join.

For embedded static SQL, Db2 for i only reuses ODPs opened by the same statement. An identical statement coded later in the program does not reuse an ODP from any other statement. If the identical statement must be run in the program many times, code it once in a subroutine and call the subroutine to run the statement.

The ODPs opened by Db2 for i are closed when any of the following occurs:

- a CLOSE, INSERT, UPDATE, DELETE, or SELECT INTO statement completes and the ODP required a temporary result that was not reusable or a subset temporary index.
- the **Reclaim Resources (RCLRSC)** command is issued. A **Reclaim Resources (RCLRSC)** is issued when the first COBOL program on the call stack ends or when a COBOL program issues the STOP RUN COBOL statement. **Reclaim Resources (RCLRSC)** does not close the ODPs created for programs precompiled using CLOSQLCSR(*ENDJOB). For interaction of **Reclaim Resources (RCLRSC)** with non-default activation groups, see the following books:

- WebSphere® Development Studio: ILE C/C++ Programmer's Guide
- WebSphere Development Studio: ILE COBOL Programmer's Guide
- WebSphere Development Studio: ILE RPG Programmer's Guide

- the last program containing SQL statements on the call stack exits. Exception is for ODPs created for programs precompiled using CLOSQLCSR(*ENDJOB) or modules precompiled using CLOSQLCSR(*ENDACTGRP).
- a CONNECT (Type 1) statement changes the application server for an activation group, all ODPs created for the activation group are closed.
- a DISCONNECT statement ends a connection to the application server, all ODPs for that application server are closed.
- a released connection is ended by a successful COMMIT, all ODPs for that application server are closed.
- the threshold for open cursors specified by the query options file (QAQQINI) parameter OPEN_CURSOR_THRESHOLD is reached.
- the SQL LOCK TABLE or CL ALCOBJ OBJ((filename *FILE *EXCL)) CONFLICT(*RQSRLS) command closes any pseudo-closed cursors associated with the specified table.
- an application has requested a close, but the data path was left open. The ODP can be forced closed for a specific file by using the ALCOBJ CL command. This close does not force the ODP to close if the application has not requested that the cursor be closed. The syntax for the command is: ALCOBJ OBJ((library/file *FILE *EXCL)) CONFLICT(*RQSRLS).
- an MQT plan expired based on the timestamp.
- an incompatible commitment control change occurred.
- the table size changed beyond tolerance. The optimizer needs to reoptimize based on the new table size.
- a new index or indexes were created. The optimizer can cost a plan created with the new indexes and compare its cost to the previous plan.
- new statistics were created. The optimizer can take advantage of these new statistics to create a more efficient plan.
- host variables are incompatible with a non-reusable MTI, an MQT, or a sparse index used to implement the query.
- data is warm (in memory).
- the OPTIMIZATION_GOAL *All IO or *First IO specified in query options file QAQQINI was changed.
- a hard close was forced.

The optimizer does not recognize that query selectivity has changed due to host variable changes. It continues to use the existing open and access plan. Change of selectivity due to host variables is only evaluated at full open time unless the PSEUDO_OPEN_CHECK_HOST_VARS qaqqini option is altered.

You can control whether the system keeps the ODPs open in the following ways:

- Design the application so a program that issues an SQL statement is always on the call stack
- Use the CLOSQLCSR(*ENDJOB) or CLOSQLCSR(*ENDACTGRP) parameter
- By specifying the OPEN_CURSOR_THRESHOLD and OPEN_CURSOR_CLOSE_COUNT parameters of the query options file (QAQQINI)

You can control whether the optimizer factors in host variable selectivity once in pseudo mode for queries with host variable that have considerable selectivity variability.

- By specifying the PSEUDO_OPEN_CHECK_HOST_VARS parameter of the query options file (QAQQINI)

An open operation occurs for the first execution of each UPDATE WHERE CURRENT OF, when any SET clause expression contains an operator or function. The open can be avoided by coding the function or operation in the host language code.

For example, the following UPDATE causes the system to do an open operation:

```
EXEC SQL
 FETCH EMPT INTO :SALARY
END-EXEC.

EXEC SQL
 UPDATE CORPDATA.EMPLOYEE
   SET SALARY = :SALARY + 1000
   WHERE CURRENT OF EMPT
END-EXEC.
```

Instead, use the following coding technique to avoid opens:

```
EXEC SQL
 FETCH EMPT  INTO  :SALARY
END EXEC.

ADD 1000 TO SALARY.

EXEC SQL
 UPDATE CORPDATA.EMPLOYEE
   SET SALARY = :SALARY
   WHERE CURRENT OF EMPT
END-EXEC.
```

You can determine whether SQL statements result in full opens in several ways. The preferred methods are to use the Database Monitor or by looking at the messages issued while debug is active. You can also use the CL commands **Trace Job (TRCJOB)** or **Display Journal (DSPJRN)**.

Related information

Reclaim Resources (RCLRSC) command

Trace Job (TRCJOB) command

Display Journal (DSPJRN) command

RPG

COBOL

C and C++

Retaining cursor positions

You can improve performance by retaining cursor positions.

Retaining cursor positions for non-ILE program calls

For non-ILE program calls, the close SQL cursor (CLOSQLCSR) parameter allows you to specify the scope of the following:

- The cursors
- The prepared statements
- The locks

When used properly, the CLOSQLCSR parameter can reduce the number of SQL OPEN, PREPARE, and LOCK statements needed. It can also simplify applications by allowing you to retain cursor positions across program calls.

***ENDPGM**
: The default for all non-ILE precompilers. With this option, a cursor remains open and accessible only while the program that opened it is on the call stack. When the program ends, the SQL cursor can no longer be used. Prepared statements are also lost when the program ends. Locks, however, remain until the last SQL program on the call stack has completed.

***ENDSQL**
: SQL cursors and prepared statements that are created by a program remain open until the last SQL program on the call stack has completed. They cannot be used by other programs, only by a different call to the same program. Locks remain until the last SQL program in the call stack completes.

***ENDJOB**
: This option allows you to keep SQL cursors, prepared statements, and locks active for the duration of the job. When the last SQL program on the stack has completed, any SQL resources created by *ENDJOB programs are still active. The locks remain in effect. The SQL cursors that were not explicitly closed by the CLOSE, COMMIT, or ROLLBACK statements remain open. The prepared statements are still usable on subsequent calls to the same program.

Related reference

Effects of precompile options on database performance

Several precompile options are available for creating SQL programs with improved performance. They are only options because using them could impact the function of the application. For this reason, the default value for these parameters is the value that ensures successful migration of applications from prior releases. However, you can improve performance by specifying other options.

Retaining cursor positions across ILE program calls

For ILE program calls, the close SQL cursor (CLOSQLCSR) parameter allows you to specify the scope of the following:

- The cursors
- The prepared statements
- The locks

When used properly, the CLOSQLCSR parameter can reduce the number of SQL OPEN, PREPARE, and LOCK statements needed. It can also simplify applications by allowing you to retain cursor positions across program calls.

***ENDACTGRP**
: The default for the ILE precompilers. With this option, SQL cursors and prepared statements remain open until the activation group that the program is running under ends. They cannot be used by other programs, only by a different call to the same program. Locks remain until the activation group ends.

***ENDMOD**
: With this option, a cursor remains open and accessible only while the module that opened it is active. When the module ends, the SQL cursor can no longer be used. Prepared statements are also lost when the module ends. Locks, however, remain until the last SQL program in the call stack completes.

General rules for retaining cursor positions for all program calls

Programs compiled with either CLOSQLCSR(*ENDPGM) or CLOSQLCSR(*ENDMOD) must open a cursor every time the program or module is called, in order to access the data. If the SQL program or module is called several times, and you want to take advantage of a reusable ODP, then the cursor must be explicitly closed before the program or module exits.

Using the CLOSQLCSR parameter and specifying *ENDSQL, *ENDJOB, or *ENDACTGRP, you might not need to run an OPEN and a CLOSE statement on every call. In addition to having fewer statements to run, you can maintain the cursor position between calls to the program or module.

The following examples of SQL statements help demonstrate the advantage of using the CLOSQLCSR parameter:

```
EXEC SQL
 DECLARE DEPTDATA CURSOR FOR
   SELECT EMPNO, LASTNAME
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = :DEPTNUM
END-EXEC.

EXEC SQL
 OPEN DEPTDATA
END-EXEC.

EXEC SQL
 FETCH DEPTDATA INTO :EMPNUM, :LNAME
END-EXEC.

EXEC SQL
```

```
     CLOSE DEPTDATA
    END-EXEC.
```

If this program is called several times from another SQL program, it is able to use a reusable ODP. This technique means that, as long as SQL remains active between the calls to this program, the OPEN statement does not require a database open operation. However, the cursor is still positioned to the first result row after each OPEN statement, and the FETCH statement will always return the first row.

In the following example, the CLOSE statement has been removed:

```
    EXEC SQL
     DECLARE DEPTDATA CURSOR FOR
       SELECT EMPNO, LASTNAME
         FROM CORPDATA.EMPLOYEE
         WHERE WORKDEPT = :DEPTNUM
    END-EXEC.

      IF CURSOR-CLOSED IS = TRUE THEN
    EXEC SQL
     OPEN DEPTDATA
    END-EXEC.

    EXEC SQL
     FETCH DEPTDATA INTO :EMPNUM, :LNAME
    END-EXEC.
```

If this program is precompiled with the *ENDJOB or *ENDACTGRP option and the activation group remains active, the cursor position is maintained. The cursor position is also maintained when the following occurs:

- The program is precompiled with the *ENDSQL option.
- SQL remains active between program calls.

The result of this strategy is that each call to the program retrieves the next row in the cursor. On subsequent data requests, the OPEN statement is unnecessary and, in fact, fails with a -502 SQLCODE. You can ignore the error, or add code to skip the OPEN. Use a FETCH statement first, and then run the OPEN statement only if the FETCH operation failed.

This technique also applies to prepared statements. A program can first try the EXECUTE, and if it fails, perform the PREPARE. The result is that the PREPARE is only needed on the first call to the program, assuming that the correct CLOSQLCSR option was chosen. If the statement can change between calls to the program, perform the PREPARE in all cases.

The main program might also control cursors by sending a special parameter on the first call only. This special parameter value indicates that because it is the first call, the subprogram performs the OPENs, PREPAREs, and LOCKs.

Note: If you are using COBOL programs, do not use the STOP RUN statement. When the first COBOL program on the call stack ends or a STOP RUN statement runs, a reclaim resource (RCLRSC) operation is done. This operation closes the SQL cursor. The *ENDSQL option does not work as you wanted.

Programming techniques for database performance

By changing the coding of your queries, you can improve their performance.

Use the OPTIMIZE clause

If an application is not going to retrieve the entire result table for a cursor, using the OPTIMIZE clause can improve performance. The query optimizer modifies the cost estimates to retrieve the subset of rows using the value specified on the OPTIMIZE clause.

Assume that the following query returns 1000 rows:

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
```

```
      WHERE WORKDEPT = 'A00'
    ORDER BY LASTNAME
    OPTIMIZE FOR 100 ROWS
END EXEC.
```

Note: The values that can be used for the preceding OPTIMIZE clause are 1–9999999 or ALL.

The optimizer calculates the following costs.

The optimize ratio = optimize for n rows value / estimated number of rows in answer set.

```
Cost using a temporarily created index:

               Cost to retrieve answer set rows
            +  Cost to create the index
            +  Cost to retrieve the rows again
                  with a temporary index        * optimize ratio

Cost using a SORT:

               Cost to retrieve answer set rows
            +  Cost for SORT input processing
            +  Cost for SORT output processing  * optimize ratio


Cost using an existing index:

               Cost to retrieve answer set rows
               using an existing index          * optimize ratio
```

In the previous examples, the estimated cost to sort or to create an index is not adjusted by the optimize ratio. This method allows the optimizer to balance the optimization and preprocessing requirements.

If the optimize number is larger than the number of rows in the result table, no adjustments are made to the cost estimates.

If the OPTIMIZE clause is not specified for a query, a default value is used based on the statement type, value of ALWCPYDTA, or output device.

Table 54. OPTIMIZE FOR n ROWS default value

Statement Type	ALWCPYDTA(*OPTIMIZE)	ALWCPYDTA(*YES or *NO)
DECLARE CURSOR	The number or rows in the result table.	30 rows or the number of rows in the result table.
Embedded Select	2	2
INTERACTIVE Select output to display	30 rows or the number of rows in the result table.	30 rows or the number of rows in the result table.
INTERACTIVE Select output to printer or database table	The number of rows in the result table.	The number of rows in the result table.

The OPTIMIZE clause influences the optimization of a query:

- To use an existing index (by specifying a small number).
- To enable the creation of an index, or run a sort or hash by specifying many possible rows in the answer set.

Related information

select-statement

Use FETCH FOR n ROWS

Applications that perform many FETCH statements in succession could be improved by using FETCH FOR n ROWS. With this clause, you can retrieve multiple rows of table data with a single FETCH, putting them into a host structure array or row storage area.

An SQL application that uses a FETCH statement without the FOR n ROWS clause can be improved by using the multiple-row FETCH statement to retrieve multiple rows. After the host structure array or row storage area is filled by the FETCH, the application loops through the data, processing each of the individual rows. The statement runs faster because the SQL run-time was called only once and all the data was simultaneously returned to the application program.

You can change the application program to allow the database manager to block the rows that the SQL run-time retrieves from the tables.

In the following table, the program attempted to FETCH 100 rows into the application. Note the differences in the table for the number of calls to SQL runtime and the database manager when blocking can be performed.

Table 55. Number of Calls Using a FETCH Statement

	Database Manager Not Using Blocking	**Database Manager Using Blocking**
Single-Row FETCH Statement	100 SQL calls 100 database calls	100 SQL calls one database call
Multiple-Row FETCH Statement	one SQL runtime call 100 database calls	one SQL runtime call one database call

Related information

FETCH statement

Improve SQL blocking performance when using FETCH FOR n ROWS

Use these performance techniques to improve SQL blocking performance when using FETCH FOR n ROWS.

You can improve SQL blocking performance with the following:

- Match the attribute information in the host structure array or the descriptor associated with the row storage area with the attributes of the columns retrieved.
- Retrieve as many rows as possible with a single multiple-row FETCH call. The blocking factor for a multiple-row FETCH request is not controlled by the system page sizes or the SEQONLY parameter on the OVRDBF command. It is controlled by the number of rows that are requested on the multiple-row FETCH request.
- Do not mix single- and multiple-row FETCH requests against the same cursor within a program. If one FETCH against a cursor is treated as a multiple-row FETCH, all fetches against that cursor are treated as multiple-row fetches. In that case, each of the single-row FETCH requests is treated as a multiple-row FETCH of one row.
- Do not use the PRIOR, CURRENT, and RELATIVE scroll options with multiple-row FETCH statements. To allow random movement of the cursor by the application, the database manager must maintain the same cursor position as the application. Therefore, the SQL run-time treats all FETCH requests against a scrollable cursor with these options specified as multiple-row FETCH requests.

Use INSERT n ROWS

Applications that perform many INSERT statements in succession could be improved by using INSERT n ROWS. With this clause, you can insert one or more rows of data from a host structure array into a target table. This array must be an array of structures where the elements of the structure correspond to columns in the target table.

An SQL application that loops over an INSERT...VALUES statement (without the n ROWS clause) can be improved by using the INSERT n ROWS statement to insert multiple rows into the table. After the application has looped to fill the host array with rows, a single INSERT n ROWS statement inserts the entire array into the table. The statement runs faster because the SQL runtime was only called once and all the data was simultaneously inserted into the target table.

In the following table, the program attempted to INSERT 100 rows into a table. Note the differences in the number of calls to SQL runtime and to the database manager when blocking can be performed.

Table 56. Number of Calls Using an INSERT Statement

	Database Manager Not Using Blocking	**Database Manager Using Blocking**
Single-Row INSERT Statement	100 SQL runtime calls 100 database calls	100 SQL runtime calls one database call
Multiple-Row INSERT Statement	1 SQL runtime call 100 database calls	1 SQL runtime call 1 database call

Related information

INSERT statement

Control database manager blocking

To improve performance, the SQL runtime attempts to retrieve and insert rows from the database manager a block at a time whenever possible.

You can control blocking, if you want. Use the SEQONLY parameter on the CL command **`Override Database File (OVRDBF)`** before calling the application program that contains the SQL statements. You can also specify the ALWBLK parameter on the CRTSQLxxx commands or use the QSY2.OVERRIDE_TABLE application service.

The database manager does not allow blocking in the following situations:

- The cursor is update or delete capable.
- The length of the row plus the feedback information is greater than 32767. The minimum size for the feedback information is 11 bytes. The feedback size is increased by the number of bytes in the index key columns used by the cursor, and the number of key columns, if any, that are null capable.
- COMMIT(*CS) is specified, and ALWBLK(*ALLREAD) is not specified.
- COMMIT(*ALL) is specified, and the following are true:
  - A SELECT INTO statement or a blocked FETCH statement is not used
  - The query does not use column functions or specify group by columns.
  - A temporary result table does not need to be created.
- COMMIT(*CHG) is specified, and ALWBLK(*ALLREAD) is not specified.
- The cursor contains at least one subquery and the outermost subselect provided a correlated reference for a subquery, or the outermost subselect processed a subquery with an IN, = ANY, or < > ALL subquery predicate operator, which is treated as a correlated reference, and that subquery is not isolatable.

The SQL runtime automatically blocks rows with the database manager in the following cases:

- INSERT

  If an INSERT statement contains a select-statement, inserted rows are blocked and not inserted into the target table until the block is full. The SQL runtime automatically does blocking for blocked inserts.

  Note: If an INSERT with VALUES is specified, the SQL runtime might not close the internal cursor used to perform the inserts until the program ends. If the same INSERT statement is run again, a full open is not necessary and the application runs much faster.
- OPEN

Blocking is done under the OPEN statement when the rows are retrieved if all the following conditions are true:

- The cursor is only used for FETCH statements.
- No EXECUTE or EXECUTE IMMEDIATE statements are in the program, or ALWBLK(*ALLREAD) was specified, or the cursor is declared with the FOR FETCH ONLY clause.
- COMMIT(*CHG) and ALWBLK(*ALLREAD) are specified, COMMIT(*CS) and ALWBLK(*ALLREAD) are specified, or COMMIT(*NONE) is specified.

Related reference

OVERRIDE_TABLE procedure
The OVERRIDE_TABLE procedure sets the blocking size for a table.

Effects of precompile options on database performance
Several precompile options are available for creating SQL programs with improved performance. They are only options because using them could impact the function of the application. For this reason, the default value for these parameters is the value that ensures successful migration of applications from prior releases. However, you can improve performance by specifying other options.

Related information

Override Database File (OVRDBF) command

Optimize the number of columns that are selected with SELECT statements

For each column in the SELECT statement, the database manager retrieves the data from the underlying table and maps it to a host variable in the application program. By minimizing the number of columns that are specified, processing unit resource usage can be conserved.

Even though it is convenient to code SELECT *, it is far better to explicitly code the columns that are required for the application. This technique is especially important for index-only access, or if all the columns participate in a sort operation (as in SELECT DISTINCT and SELECT UNION).

This technique is also important when considering index only access. You minimize the number of columns in a query and increase the odds that an index can be used to completely satisfy the data request.

Related information

select-statement

Eliminate redundant validation with SQL PREPARE statements

The processing which occurs when an SQL PREPARE statement is run is like the processing which occurs during precompile processing.

The following processing occurs for the statement that is being prepared:

- The syntax is checked.
- The statement is validated to ensure that the usage of objects is valid.
- An access plan is built.

Again when the statement is executed or opened, the database manager revalidates that the access plan is still valid. Much of this open processing validation is redundant with the validation which occurred during the PREPARE processing. The DLYPRP(*YES) parameter specifies whether PREPARE statements in this program completely validates the dynamic statement. The validation is completed when the dynamic statement is opened or executed. This parameter can provide a significant performance enhancement for programs which use the PREPARE SQL statement because it eliminates redundant validation. Programs that specify this precompile option must check the SQLCODE and SQLSTATE after running the OPEN or EXECUTE statement to ensure that the statement is valid. DLYPRP(*YES) does not provide any performance improvement if the INTO clause is used on the PREPARE statement, or if a DESCRIBE statement uses the dynamic statement before an OPEN is issued for the statement.

Related reference

Effects of precompile options on database performance

Several precompile options are available for creating SQL programs with improved performance. They are only options because using them could impact the function of the application. For this reason, the default value for these parameters is the value that ensures successful migration of applications from prior releases. However, you can improve performance by specifying other options.

Related information

Prepare statement

Page interactively displayed data with REFRESH(*FORWARD)

In large tables, paging performance is typically degraded because of the REFRESH(*ALWAYS) parameter on the `Start SQL (STRSQL)` command. STRSQL dynamically retrieves the latest data directly from the table. Paging performance can be improved by specifying REFRESH(*FORWARD).

When interactively displaying data using REFRESH(*FORWARD), the results of a select-statement are copied to a temporary table as you page forward through the display. Other users sharing the table can change the rows while you are displaying the select-statement results. If you page backward or forward to rows that have already been displayed, the rows shown are in the temporary table instead of the updated table.

The refresh option can be changed on the Session Services display.

Related information

Start SQL (STRSQL) command

Improve concurrency by avoiding lock waits

The concurrent access resolution option directs the database manager on how to handle cases of record lock conflicts under certain isolation levels.

The concurrent access resolution, when applicable, can have one of the following values:

- **Wait for outcome** (default). This value directs the database manager to wait for the commit or rollback when encountering locked data in the process of being updated or deleted. Locked rows that are in the process of being inserted are not skipped. This option does not apply for read-only queries running under isolation level None or Uncommitted Read.
- **Use currently committed.** This value allows the database manager to use the currently committed version of the data for read-only queries when encountering locked data in the process of being updated or deleted. Locked rows in the process of being inserted can be skipped. This option applies if possible when the isolation level in effect is Cursor Stability and is ignored otherwise.
- **Skip locked data.** This value directs the database manager to skip rows in the case of record lock conflicts. This option is applicable only when the query is running under an isolation level of Cursor Stability or Read Stability and additionally for UPDATE and DELETE queries when the isolation level is None or Uncommitted Read.

The concurrent access resolution values of USE CURRENTLY COMMITTED and SKIP LOCKED DATA can be used to improve concurrency by avoiding lock waits. However, care must be used when using these options because they might affect application functionality. For more information on the USE CURRENTLY COMMITTED option, see Concurrency.

WAIT FOR OUTCOME, USE CURRENTLY COMMITTED, and SKIP LOCKED DATA can be specified as the concurrent-access-resolution-clause in the attribute-string of a PREPARE statement.

Additionally, they can be specified as the concurrent-access-resolution-clause at the statement level on a select-statement, SELECT INTO, searched UPDATE, or searched DELETE statement.

Concurrent access resolution is also specifiable as a precompiler option by using the CONACC parameter on the CRTSQLxxx. The CONACC parameter accepts one of the following values:

- *DFT - specifies that the concurrent access option is not explicitly set for this program. The value that is in effect when the program is invoked is used. The value can be set using the SQL_CONCURRENT_ACCESS_RESOLUTION option in the query options file QAQQINI.
- *CURCMT - use currently committed.
- *WAIT - wait for outcome.

These same options can be set on the RUNSQLSTM and RUNSQL CL commands and by using the SET OPTION SQL statement. Concurrent access resolution can be specified for SQL triggers, functions, and procedures by using the SET OPTION statement.

When the concurrent access resolution option is not directly set by the application, it is set to the value of the SQL_CONCURRENT_ACCESS_RESOLUTION option in the query options file QAQQINI. This option accepts one of the following values:

- *DEFAULT - the default value is set to *WAIT.
- *CURCMT - use currently committed.
- *WAIT - wait for outcome.

Related reference

QAQQINI query options
There are different options available for parameters in the QAQQINI file.

Related information

concurrent-access-resolution-clause

Concurrency

General Db2 for i performance considerations

As you code your applications, there are some general tips that can help you optimize performance.

Effects on database performance when using long object names

Long object names are converted internally to system object names when used in SQL statements. This conversion can have some performance impacts. Names of tables, views, indexes, and aliases that are 30 characters or less will generally perform much better names longer than 30 characters.

Qualify the long object name with a library name and the conversion to the short name happens at precompile time. In this case, there is minimal performance impact when the statement is executed. Otherwise, the conversion is done at execution time and has a small performance impact.

Effects of precompile options on database performance

Several precompile options are available for creating SQL programs with improved performance. They are only options because using them could impact the function of the application. For this reason, the default value for these parameters is the value that ensures successful migration of applications from prior releases. However, you can improve performance by specifying other options.

The following table shows these precompile options and their performance impacts.

Some of these options might be suitable for most of your applications. Use the command **CRTDUPOBJ** to create a copy of the SQL **CRTSQLxxx** command. and the **CHGCMDDFT** command to customize the optimal values for the precompile parameters. The **DSPPGM**, **DSPSRVPGM**, **DSPMOD**, or **PRTSQLINF** commands can be used to show the precompile options that are used for an existing program object.

Table 57. Precompile options and their performance impacts

Precompile Option	Optimal Value	Improvements	Considerations
ALWCPYDTA	*OPTIMIZE (the default)	Queries where the ordering or grouping criteria conflicts with the selection criteria.	A copy of the data could be made when the query is opened.
ALWBLK	*ALLREAD (the default)	Additional read-only cursors use blocking.	ROLLBACK HOLD might not change the position of a read-only cursor. Dynamic processing of positioned updates or deletes might fail.
CLOSQLCSR	*ENDJOB, *ENDSQL, or *ENDACTGRP	Cursor position can be retained across program invocations.	Implicit closing of SQL cursor is not done when the program invocation ends.
DLYPRP	*YES	Programs using SQL PREPARE statements could run faster.	Complete validation of the prepared statement is delayed until the statement is run or opened.
TGTRLS	*CURRENT (the default)	The precompiler can generate code that takes advantage of performance enhancements available in the current release.	The program object cannot be used on a system from a previous release.

Related reference

Effects of the ALWCPYDTA parameter on database performance
Some complex queries can perform better by using a sort or hashing method to evaluate the query instead of using or creating an index.

Control database manager blocking
To improve performance, the SQL runtime attempts to retrieve and insert rows from the database manager a block at a time whenever possible.

Retaining cursor positions for non-ILE program calls
For non-ILE program calls, the close SQL cursor (CLOSQLCSR) parameter allows you to specify the scope of the following:

Eliminate redundant validation with SQL PREPARE statements
The processing which occurs when an SQL PREPARE statement is run is like the processing which occurs during precompile processing.

Effects of the ALWCPYDTA parameter on database performance

Some complex queries can perform better by using a sort or hashing method to evaluate the query instead of using or creating an index.

By using the sort or hash, the database manager is able to separate the row selection from the ordering and grouping process. Bitmap processing can also be partially controlled through this parameter. This separation allows the use of the most efficient index for the selection. For example, consider the following SQL statement:

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
```

```
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'A00'
     ORDER BY LASTNAME
END-EXEC.
```

The above SQL statement can be written in the following way by using the OPNQRYF command:

```
OPNQRYF FILE(CORPDATA/EMPLOYEE)
        FORMAT(FORMAT1)
        QRYSLT(WORKDEPT *EQ ''A00'')
        KEYFLD(LASTNAME)
```

In the preceding example, when ALWCPYDTA(*NO) or ALWCPYDTA(*YES) is specified, the database manager could try to create an index from the first index with a column named LASTNAME, if such an index exists. The rows in the table are scanned, using the index, to select only the rows matching the WHERE condition.

If ALWCPYDTA(*OPTIMIZE) is specified, the database manager uses an index with the first index column of WORKDEPT. It then makes a copy of all the rows that match the WHERE condition. Finally, it could sort the copied rows by the values in LASTNAME. This row selection processing is more efficient, because the index used immediately locates the rows to be selected.

ALWCPYDTA(*OPTIMIZE) optimizes the total time that is required to process the query. However, the time required to receive the first row could be increased because a copy of the data must be made before returning the first row of the result table. This initial change in response time could be important for applications that are presenting interactive displays or that retrieve only the first few rows of the query. The Db2 for i query optimizer can be influenced to avoid sorting by using the OPTIMIZE clause.

Queries that involve a join operation might also benefit from ALWCPYDTA(*OPTIMIZE) because the join order can be optimized regardless of the ORDER BY specification.

Related concepts

Plan cache
The plan cache is a repository that contains the access plans for queries that were optimized by SQE.

Related reference

Effects of precompile options on database performance
Several precompile options are available for creating SQL programs with improved performance. They are only options because using them could impact the function of the application. For this reason, the default value for these parameters is the value that ensures successful migration of applications from prior releases. However, you can improve performance by specifying other options.

Radix index scan
A radix index scan operation is used to retrieve the rows from a table in a keyed sequence. Like a table scan, all the rows in the index are sequentially processed, but the resulting row numbers are sequenced based upon the key columns.

Radix index probe
A radix index probe operation is used to retrieve the rows from a table in a keyed sequence. The main difference between the radix index probe and the scan is that the rows returned are first identified by a probe operation to subset them.

Tips for using VARCHAR and VARGRAPHIC data types in databases

Variable-length column (VARCHAR or VARGRAPHIC) support allows you to define any number of columns in a table as variable length. If you use VARCHAR or VARGRAPHIC support, the size of a table can typically be reduced.

Data in a variable-length column is stored internally in two areas: a fixed-length or ALLOCATE area and an overflow area. If a default value is specified, the allocated length is at least as large as the value. The following points help you determine the best way to use your storage area.

When you define a table with variable-length data, you must decide the width of the ALLOCATE area. If the primary goal is:

- **Space saving:** use ALLOCATE(0).
- **Performance:** the ALLOCATE area must be wide enough to incorporate at least 90% to 95% of the values for the column.

It is possible to balance space savings and performance. In the following example of an electronic telephone book, the following data is used:

- 8600 names that are identified by: last, first, and middle name
- The Last, First, and Middle columns are variable length.
- The shortest last name is two characters; the longest is 22 characters.

This example shows how space can be saved by using variable-length columns. The fixed-length column table uses the most space. The table with the carefully calculated allocate sizes uses less disk space. The table that was defined with no allocate size (with all the data stored in the overflow area) uses the least disk space.

Table 58. Disk space used with variable-length columns

Variety of Support	Last Name Max/Alloc	First Name Max/Alloc	Middle Name Max/Alloc	Total Physical File Size	Number of Rows in Overflow Space
Fixed Length	22	22	22	567 K	0
Variable Length	40/10	40/10	40/7	408 K	73
Variable-Length Default	40/0	40/0	40/0	373 K	8600

In many applications, performance must be considered. If you use the default ALLOCATE(0), it doubles the disk unit traffic. ALLOCATE(0) requires two reads; one to read the fixed-length portion of the row and one to read the overflow space. The variable-length implementation, with the carefully chosen ALLOCATE, minimizes overflow and space and maximizes performance. The size of the table is 28% smaller than the fixed-length implementation. Because 1% of rows are in the overflow area, the access requiring two reads is minimized. The variable-length implementation performs about the same as the fixed-length implementation.

To create the table using the ALLOCATE keyword:

```
CREATE TABLE PHONEDIR
         (LAST    VARCHAR(40) ALLOCATE(10),
          FIRST   VARCHAR(40) ALLOCATE(10),
          MIDDLE  VARCHAR(40) ALLOCATE(7))
```

If you are using host variables to insert or update variable-length columns, use variable length host variables. Because blanks are not truncated from fixed-length host variables, using fixed-length host variables can cause more rows to spill into the overflow space. This increases the size of the table.

In this example, fixed-length host variables are used to insert a row into a table:

```
01  LAST-NAME PIC X(40).
   …
    MOVE "SMITH" TO LAST-NAME.
    EXEC SQL
      INSERT INTO PHONEDIR
       VALUES(:LAST-NAME, :FIRST-NAME, :MIDDLE-NAME, :PHONE)
    END-EXEC.
```

The host-variable LAST-NAME is not variable length. The string “SMITH”, followed by 35 blanks, is inserted into the VARCHAR column LAST. The value is longer than the allocate size of 10. 30 of 35 trailing blanks are in the overflow area.

In this example, variable-length host variables are used to insert a row into a table:

```
01  VLAST-NAME.
    49 LAST-NAME-LEN PIC S9(4) BINARY.
    49 LAST-NAME-DATA PIC X(40).
   …
    MOVE "SMITH" TO LAST-NAME-DATA.
    MOVE 5 TO LAST-NAME-LEN.
    EXEC SQL
      INSERT INTO PHONEDIR
     VALUES(:VLAST-NAME, :VFIRST-NAME, :VMIDDLE-NAME, :PHONE)
    END-EXEC.
```

The host variable VLAST-NAME is variable length. The actual length of the data is set to 5. The value is shorter than the allocated length. It can be placed in the fixed portion of the column.

Running the **Reorganize Physical File Member (RGZPFM)** command against tables that contain variable-length columns can improve performance. The fragments in the overflow area that are not in use are compacted by the **Reorganize Physical File Member (RGZPFM)** command. This technique reduces the read time for rows that overflow, increases the locality of reference, and produces optimal order for serial batch processing.

Choose the appropriate maximum length for variable-length columns. Selecting lengths that are too long increases the process access group (PAG). A large PAG slows performance. A large maximum length makes SEQONLY(*YES) less effective. Variable-length columns longer than 2000 bytes are not eligible as key columns.

Using LOBs and VARCHAR in the same table

Storage for LOB columns allocated in the same manner as VARCHAR columns. When a column stored in the overflow storage area is referenced, currently all the columns in that area are paged into memory. A reference to a "smaller" VARCHAR column that is in the overflow area can potentially force extra paging of LOB columns. For example, A VARCHAR(256) column retrieved by application has side-effect of paging in two 5 MB BLOB columns that are in the same row. In order to prevent this side-effect, you might want to use ALLOCATE keyword to ensure that only LOB columns are stored in the overflow area.

Related information

Reorganize Physical File Member (RGZPFM) command

Reorganizing a physical file

Embedded SQL programming

Using field procedures to provide column level encryption

Field procedures can provide column level encryption in DB2 for i.

A field procedure is a user-written exit routine to transform values in a single column. When values in the column are changed, or new values inserted, the field procedure is invoked for each value. The field procedure can transform that value (encode it) in any way. The encoded value is then stored. When values are retrieved from the column, the field procedure is invoked for each encoded value. The field procedure decodes each value back to the original value. Any indexes defined on a column that uses a field procedure are built with encoded values.

Field procedures are assigned to a table by the FIELDPROC clause of CREATE TABLE and ALTER TABLE.

A field procedure that is specified for a column is invoked in three general situations:

- For field-definition, when the CREATE TABLE or ALTER TABLE statement that names the procedure is executed. During this invocation, the procedure is expected to:
  - Determine whether the data type and attributes of the column are valid.
  - Verify the literal list, and change it if wanted.
  - Provide the field description of the column.
- For field-encoding, when a column value is field-encoded. That occurs for any value that:

- is inserted in the column by an SQL INSERT statement, SQL MERGE statement, or native write.
- is changed by an SQL UPDATE statement, SQL MERGE statement, or native update.
- is the target column for a copy from a column with an associated field procedure. The field procedure might be invoked to encode the copied data. Examples include SQL Statements ALTER TABLE or CREATE TABLE LIKE/AS and CL commands CPYF and RGZPFM.
- is compared to a column with a field procedure. The QAQQINI option FIELDPROC_ENCODED_COMPARISON is used to determine if the column value is decoded, or the host variable, constant, or join column is encoded.
- is the DEFAULT value for a column with an associated field procedure in a CREATE or ALTER TABLE statement.

If there are any **after** or **read** triggers, the field procedure is invoked *before* any of these triggers. If there are any **before** triggers, the field procedure is invoked *after* the before trigger.

- For field-decoding, when a stored value is field-decoded back into its original value. Field-decoding occurs for any value that:
  - is retrieved by an SQL SELECT or FETCH statement, or by a native read.
  - is a column with an associated field procedure that is copied. The field procedure might be invoked to decode the data before making the copy. Examples include SQL Statements ALTER TABLE, CREATE TABLE LIKE/AS, and CL commands CPYF and RGZPFM.
  - is compared to a column with a field procedure. The QAQQINI option FIELDPROC_ENCODED_COMPARISON is used by the optimizer to decide if the column value is decoded, or if the host variable or constant is encoded.

  A field procedure is never invoked to process a null value. It is also not invoked for a DELETE operation without a WHERE clause when the table has no DELETE triggers. The field procedure is invoked for empty strings.

Improving performance

For queries that use field procedures, the path length is longer due to the additional processing of calling the field procedure. In order to improve performance of queries, the SQE optimizer:

- attempts to remove decoding operations, based on the QAQQINI FIELDPROC_ENCODED COMPARISON setting.
- matches existing indexes over columns that have an associated field procedure.
- creates and uses MTIs over columns with field procedures.
- creates statistics over the encoded values through statistics collection.

The SQE optimizer attempts to do the following optimizations:

- optimization of predicates that compare a field procedure column to a constant or host variable. For example, predicate FP1(4, C1) = ‘abc' is optimized as C1 = FP1(0,‘abc'). With this specific example, the optimization is done as long as the QAQQINI option is not *NONE.
- remove field procedure decoding operations from join predicates when the same field procedure is applied to both sides of the join predicate, and no compatibility conversion is required. For example, join predicate FP1(4,T1.C1) > FP1(4,T2.C1) is rewritten as T1.C1 > T2.C1. With this specific example, the optimization is done as long as the QAQQINI option is either *ALLOW_RANGE or *ALL. This technique is also applied to = predicates when the QAQQINI option is *ALLOW_EQUAL.
- remove field procedures from GROUP BY and ORDER BY clauses. For example, ORDER BY FP1(4,C1) is rewritten as ORDER BY C1 if the QAQQINI setting is either *ALLOW_RANGE or *ALL

The CQE optimizer does not look at the QAQQINI option, which means it always runs in *NONE mode. *NONE mode requires that all references to the column are decoded before any other operation is performed. A CQE query does not match any existing indexes when the column has an associated field procedure. If an index is required, a temporary index is built with the index keys decoded.

Related reference

QAQQINI query options
There are different options available for parameters in the QAQQINI file.

Related information

Defining field procedures

CREATE TABLE

Field procedure examples

The following examples show various field procedure-related optimizations done by the SQE optimizer.

The examples show the FieldProc name along with the encoding (field procedure function code 0) or decoding (field procedure function code 4) in the pseudo-SQL. These codes indicate how the optimizer is optimizing the field procedure calls.

Given the following table and index:

```
CREATE TABLE T1 (col1 CHAR(10), col2 CHAR(10) FIELDPROC ‘FP1')
CREATE INDEX IX1 on T1(col2)
```

Example 1

A user query written as:

```
SELECT col1, col2 FROM T1 WHERE col2 = ‘abc'
```

Is represented by the optimizer as:

```
SELECT col1, FP1(4, col2)  FROM T1 WHERE FP1(4,col2) = ‘abc'
```

Note the FP1 with the decode operation around the COL2 references in the SELECT list and the WHERE clause.

Assuming the QAQQINI FIELDPROC_ENCODED COMPARISON is set to *ALLOW_EQUAL, *ALLOW_RANGE or *ALL:

The query optimizer rewrites the query as:

```
SELECT col1, ‘abc' FROM T1 WHERE col2 = FP1(0, ‘abc')
```

This rewrite allows the query optimizer to use the encoded index IX1 to implement the WHERE clause and only cause one invocation of the field procedure for the query.

Example 2

```
SELECT col2 FROM T1 ORDER BY col2
```

Is represented by the query optimizer as:

```
SELECT FP1(4, col2) FROM T1 ORDER BY FP1(4, col2)
```

The optimized version removes the FieldProc from the ORDER BY clause assuming that the field procedure QAQQINI option is set to *ALLOW_RANGE or *ALL:

```
SELECT FP1(4, col2) FROM T1 ORDER BY col2
```

Example 3

```
Select col2, COUNT(*) FROM T1 GROUP BY col2
```

Is represented by the query optimizer as:

```
Select FP1(4, col2), COUNT(*) FROM T1 GROUP BY FP1(4, col2)
```

The optimized version removes the field procedure invocation from the GROUP BY clause column col2, allowing it to group the encoded data and only run the field procedure once per group. The decoded grouped data is returned to the user. This optimization is done if the field procedure QAQQINI option is set to *ALLOW_RANGE or *ALL:

```
SELECT  FP1(4, col2), COUNT(*) FROM T1 GROUP BY col2
```

IS NULL/IS NOT NULL predicate does not require calling the field procedure field-decode option 4. The field procedure cannot change the nullability of the field.

DB2 for i Services

There are many system-provided views, procedures, and functions.

These are grouped in the following categories.

Application Services

These procedures provide interfaces that are useful for application development.

DELIMIT_NAME scalar function

The DELIMIT_NAME function returns a name with delimiters if the delimiters are needed for use in an SQL statement.

▸▸— DELIMIT_NAME — (— *name* —) —▸◂

The schema is QSYS2.

name
: A character or graphic string expression that identifies a name. The string must contain only characters allowed in an SQL identifier. If the string is longer that 128 characters, it will be truncated to 128 characters.

The result of the function is a varying length character string that contains *name* correctly delimited. This includes delimiting reserved words. If *name* is the null value or an empty string, null is returned.

Example

- Delimit these names:

```
 VALUES DELIMIT_NAME('ABC'),
        DELIMIT_NAME('abc'),
        DELIMIT_NAME('test"name'),
        DELIMIT_NAME('test''name2'),
        DELIMIT_NAME('NEW')
```

  Returns the values:

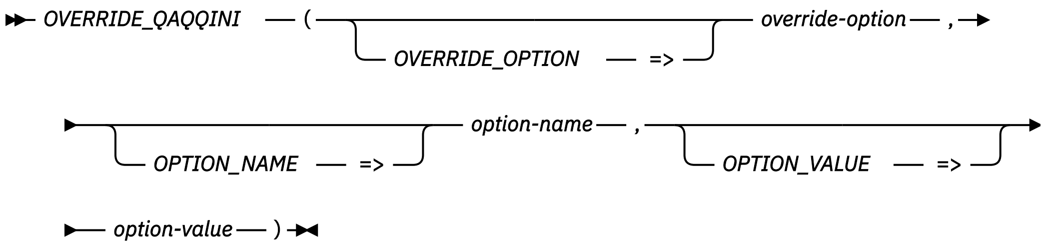
```
 ABC
"abc"
"test""name"
"test'name2"
"NEW"
```

OVERRIDE_QAQQINI procedure

The OVERRIDE_QAQQINI procedure creates and modifies a temporary version of the QAQQINI file.

The temporary QAQQINI file will be created in QTEMP. It inherits all query options that are already in place for the job. The OVERRIDE_QAQQINI procedure can be called multiple times to establish job specific QAQQINI settings.

The procedure can also be called to discard the temporary customization settings.



The schema is QSYS2.

Authorization: For most QAQQINI options, none is required. For the following three options, *JOBCTL or QIBM_DB_SQLADM function usage is required. These options are more restrictive because they can affect the performance of other jobs.

- QUERY_TIME_LIMIT when the *option-value* is not 0.
- STORAGE_LIMIT
- PARALLEL_DEGREE

override-option
: An integer value that indicates the function to perform.

1
: Create the QAQQINI override file. A procedure call with this *override-option* value must be run before option 2 can be used to change QAQQINI options.

2
: Set a QAQQINI option to the specified value. See "QAQQINI query options" on page 176 for the list of options and values.

3
: Discard the temporary QAQQINI file.

option-name
: A character or graphic string expression that identifies the name of the QAQQINI option to be changed.

option-value
: A character or graphic string expression that identifies the value to assign to the QAQQINI option identified by *option-name*.

Example

- Establish the temporary override for QAQQINI. The job's current QAQQINI values will be used as the initial values.

```
CALL QSYS2.OVERRIDE_QAQQINI(1, '', '');
```

- Temporarily override three QAQQINI values.

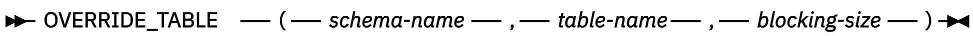
```
-- Avoid UDF timeout
CALL QSYS2.OVERRIDE_QAQQINI(2, 'UDF_TIME_OUT', '*MAX');
-- Force full opens of cursors
CALL QSYS2.OVERRIDE_QAQQINI(2, 'OPEN_CURSOR_THRESHOLD', '-1');
-- Force any saved access plans to be rebuilt
CALL QSYS2.OVERRIDE_QAQQINI(2, 'REBUILD_ACCESS_PLAN', '*YES');
```

- Discard the temporary QAQQINI file and revert to using the job's version of the QAQQINI file.

```
CALL QSYS2.OVERRIDE_QAQQINI(3, '', '');
```

OVERRIDE_TABLE procedure

The OVERRIDE_TABLE procedure sets the blocking size for a table.



The schema is QSYS2.

schema-name
A character string expression containing the name of the schema.

table-name
A character string expression containing the name of the table.

blocking-size
A character string expression containing the blocking size. It can be a specific byte count or a special value of *BUF32KB, *BUF64KB, *BUF128KB, or *BUF256KB.

Example

- Override the EMPLOYEE table to use 256K blocking for sequential processing.

```
CALL QSYS2.OVERRIDE_TABLE('CORPDATA', 'EMP', '*BUF256KB');
```

- Remove the override.

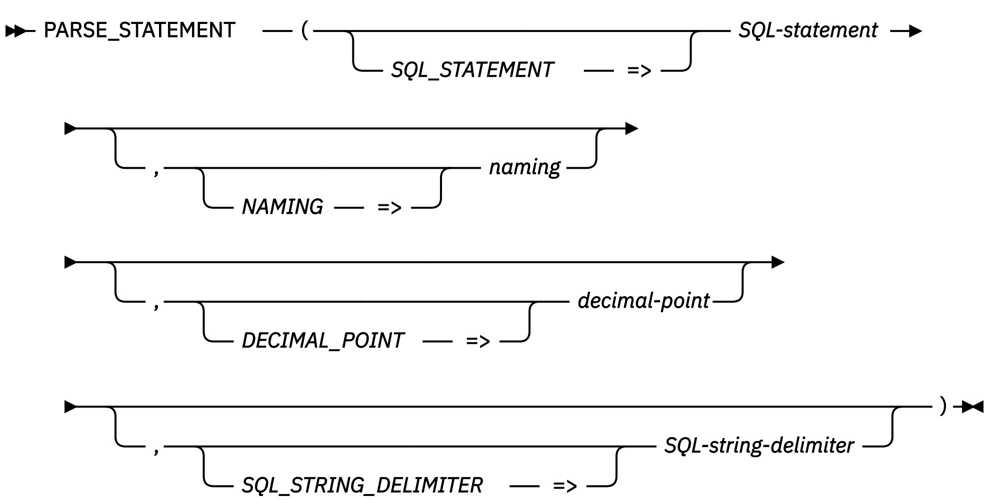
```
CALL QSYS2.OVERRIDE_TABLE('CORPDATA', 'EMP', 0);
```

Related reference

Control database manager blocking
To improve performance, the SQL runtime attempts to retrieve and insert rows from the database manager a block at a time whenever possible.

PARSE_STATEMENT table function

The PARSE_STATEMENT table function returns a list of object and column names that are used in an SQL query, data change statement, or other statement where a query or expression is specified.



The schema is QSYS2.

SQL-statement
A character or graphic string expression that contains a valid SQL statement. The maximum string length is 2 megabytes.

naming

A character or graphic string expression that defines the naming rule for the statement.

***SYS**

System naming rules apply. This is the default.

***SQL**

SQL naming rules apply.

decimal-point

A character or graphic string expression that defines the decimal point for numeric constants in *SQL-statement*.

***PERIOD or .**

The decimal point is the period. This is the default.

***COMMA or ,**

The decimal point is the comma.

SQL-string-delimiter

A character or graphic string expression that defines the string delimiter for strings in *SQL-statement*. Delimited identifiers in the SQL statement will use the opposite character.

***APOSTSQL or '**

The apostrophe character (') is used to delimit strings. This is the default.

***QUOTESQL or "**

The quote character (") is used to delimit strings.

Authorization:

- None required.

When the SQL statement is parsed, object names are identified and a result row is returned for every name. This is done at the SQL parser level where names are identified strictly by where they appear in the syntax. The following restrictions apply:

- Names used in data change statements and in any query construct are returned.
- For DDL statements, the following additional items are returned:
  - For CREATE INDEX, the table on which the index is being created.
  - For CREATE TABLE, any table referenced using the LIKE clause.
  - For CREATE TRIGGER, the table or view on which the trigger is being defined.

  DDL statements that do not contain these constructs, a query, or an expression return no rows.
- Names in a *routine-body*, *triggered-action*, and *trigger-body* are not returned. To see these references, use QSYS2.SYSPROGRAMSTMTSTAT to find all the statements for the generated program or service program and pass each of them as an argument to this table function.
- If the SQL statement is the null value, an empty string, a string of all blanks, or contains a syntax error, no row is returned.

The result of the function is a table containing a row for each name reference with the format shown in the following table. All the columns are null capable.

Table 59. PARSE_STATEMENT table function

Column Name	Data Type	Description
NAME_TYPE	VARCHAR(8)	Type of object name. **COLUMN** This is a column name or a global variable name. **FUNCTION** This is a function name. **SEQUENCE** This is a sequence name. **TABLE** This is a table, view, or alias name. **TYPE** This is a user-defined type name.
NAME	VARCHAR(128)	The object name. Contains null if NAME_TYPE is COLUMN without a table qualifier.
SCHEMA	VARCHAR(128)	The schema name. Contains null if NAME is not qualified with a schema name.
RDB	VARCHAR(128)	The relational database name. Contains null if NAME is not qualified with a relational database name.
COLUMN_NAME	VARCHAR(128)	The column name. Contains null if NAME_TYPE is not COLUMN.
USAGE_TYPE	VARCHAR(17)	How this name is used in the statement. **DDL SOURCE OBJECT** Name identifies the table an index or trigger is being created on, or the table referenced by CREATE TABLE LIKE. **EXPRESSION** Name is referenced in an index key expression. **PARAMETER DEFAULT** Name is referenced in a parameter default expression. **QUERY** Name is referenced as part of a query construct. **TARGET TABLE** This is the table that will be affected for an insert, update, delete, or merge statement. Also set for any explicitly specified columns from the target table for insert, update, and merge.
NAME_START_POSITION	INTEGER	Position within the *SQL-statement* string that this name begins. For qualified TABLE names, this is the position where the RDB or schema name begins. For all other name types, this is the position of the name.

Table 59. PARSE_STATEMENT table function (continued)

Column Name	Data Type	Description
SQL_STATEMENT_TYPE	VARCHAR(32)	Type of SQL statement. • ALTER FUNCTION • ALTER PROCEDURE • ALTER TABLE • CALL • CREATE FUNCTION • CREATE INDEX • CREATE MASK • CREATE PERMISSION • CREATE PROCEDURE • CREATE TABLE • CREATE TRIGGER • CREATE VARIABLE • CREATE VIEW • DECLARE CURSOR • DECLARE GLOBAL TEMPORARY TABLE • DELETE • EXECUTE IMMEDIATE • INSERT • MERGE • PREPARE • QUERY • SET • SET CURRENT TEMPORAL SYSTEM_TIME • UPDATE • VALUES INTO

Example

For every program and service program in library APPLIB, find all the references to table names referenced in static SQL statements.

```
WITH program_statements(naming_mode, dec_point, string_delim, stmt_text,
    system_program_name, program_type)
    AS (SELECT a.naming, a.decimal_point, a.sql_string_delimiter, b.statement_text,
               a.system_program_name, a.program_type
          FROM qsys2.sysprogramstat a INNER JOIN
               qsys2.sysprogramstmtstat b ON a.program_schema = b.program_schema AND
                                             a.program_name = b.program_name AND
                                             a.module_name = b.module_name
          WHERE a.program_schema = 'APPLIB' AND b.program_schema = 'APPLIB')
SELECT system_program_name, program_type, c.schema, c.name, stmt_text
    FROM program_statements,
    TABLE(qsys2.parse_statement(stmt_text, naming_mode, dec_point, string_delim)) c
    WHERE c.name_type = 'TABLE'
    ORDER BY c.schema, c.name;
```

WLM_SET_CLIENT_INFO procedure

The WLM_SET_CLIENT_INFO procedure sets values for the SQL client special registers.

```
►► WLM_SET_CLIENT_INFO ── ( ── client_userid ── , ── client_wrkstnname ── , ── client_applname ──►

►── , ── client_acctng ── , ── client_programid ── ) ─►◄
```

The schema is SYSPROC.

client_userid
A character string containing the value to set for the CLIENT USERID special register for the current connection.

client_wrkstnname
A character string containing the value to set for the CLIENT WRKSTNNAME special register for the current connection.

client_applname
A character string containing the value to set for the CLIENT APPLNAME special register for the current connection.

client_acctng
A character string containing the value to set for the CLIENT ACCTNG special register for the current connection.

client_programid
A character string containing the value to set for the CLIENT PROGRAMID special register for the current connection.

Performance Services

These services include procedures that provide interfaces to work with indexes and a view to see information about database monitors.

Related reference

QAQQINI file override support
If you find working with the QAQQINI query options file cumbersome, consider using the QSYS2.OVERRIDE_QAQQINI procedure. Instead of creating, managing, and using a QAQQINI *FILE object directly, this procedure can be called to work with a temporary version of the INI file. It uses user-specified options and values. The support relies upon the QTEMP library, so any changes affect only the job which calls the procedure.

ACT_ON_INDEX_ADVICE procedure

The ACT_ON_INDEX_ADVICE procedure creates new indexes for a table based on indexes that have been advised for the table.

```
▶▶─ ACT_ON_INDEX_ADVICE ── ( ── schema-name ── , ── table-name ── , ─▶

▶── times_advised ── , ── mti_used ── , ── average_estimate ── ) ─▶◀
```

The schema is SYSTOOLS.

schema-name
A character string containing the system name of the schema containing the table.

table-name
A character string containing the system name of the table. If NULL is passed, this parameter is not used to select the target index advice.

times-advised
The number of times an index should have been advised before creating a permanent index. If NULL is passed, this parameter is not used to select the target index advice.

mti-used
The number of times a maintained temporary index (MTI) has been used because a matching permanent index did not exist. If NULL is passed, this parameter is not used to select the target index advice.

average-estimate
The average estimated number of seconds needed to execute the query that drove the index advice. If NULL is passed, this parameter is not used to select the target index advice.

For each potential index meeting the specified criteria, a CREATE INDEX statement will be run to generate the permanent index. A radix index will be named `name_RADIX_INDEX_n`. An EVI index will be named `name_EVI_INDEX_n`. The *name* represents the table name and *n* is a unique number. The row containing this advised index is removed from the QSYS2.SYSIXADV table.

Examples

For schema `PRODLIB`, find all instances of index advice where a maintained temporary index was used more than 1000 times and create permanent SQL indexes.

```
CALL SYSTOOLS.ACT_ON_INDEX_ADVICE('PRODLIB',NULL,NULL,1000,NULL)
```

Related reference

SYSTOOLS

SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

DATABASE_MONITOR_INFO view

The DATABASE_MONITOR_INFO view returns information about database monitors and plan cache event monitors on the server. Database monitors are started using the Start Database Monitor (STRDBMON) command. The QSYS2.START_PLAN_CACHE_EVENT_MONITOR procedure is used to start a plan cache event monitor. SQL Performance Monitors within IBM i Navigator are synonymous with database monitors and are included in this view.

The following table describes the columns in the view. The schema is QSYS2.

Table 60. DATABASE_MONITOR_INFO view

Column Name	System Column Name	Data Type	Description
MONITOR_ID	MONITOR_ID	CHAR(10)	The system-assigned monitor ID for this monitor.
MONITOR_TYPE	MONTYPE	VARCHAR(7)	Type of monitor. **PUBLIC** A monitor is considered public when the STRDBMON JOB parameter indicates that jobs other than the current job should be monitored. Public monitors remain active until they are explicitly ended using the End Database Monitor (ENDDBMON) command. **PRIVATE** A private monitor occurs when the STRDBMON JOB parameter indicates to monitor only the current job. The monitor is ended as part of job termination processing, if needed. Only a private monitor that is active in the current connection will be returned. **EVENT** An SQL plan cache event monitor intercepts plans as they are moved from the plan cache into a database monitor file.
MONITOR_STATUS	STATUS	VARCHAR(8)	Status of this monitor. **ACTIVE** Monitor is active. **INACTIVE** For a PUBLIC or PRIVATE monitor, it is inactive and can become ACTIVE. For an EVENT monitor, entries are no longer being collected. **CLOSING** The PUBLIC or PRIVATE monitor is not active or is in the processing of ending. It is not known if the entry can be reused for monitoring.
MONITOR_RECORD_TYPE	RCDTYPE	VARCHAR(6)	Type of database records in this monitor. **DETAIL** Both basic and detail database monitor records. An EVENT monitor always has a value of DETAIL. **BASIC** Only basic database monitor records
MONITOR_LIBRARY	MONLIB	VARCHAR(10)	Library for this monitor.
MONITOR_FILE	MONFILE	VARCHAR(10)	The file to which the database activity detail is written for this monitor.

Table 60. DATABASE_MONITOR_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MONITOR_MEMBER	MONMBR	VARCHAR(10)	Member for this monitor.
IASP_NUMBER	IASPNUMBER	SMALLINT	The independent auxiliary storage pool (IASP) number for the monitor file.
MONITOR_MEMBER_OPTION	MBROPT	VARCHAR(7) Nullable	Value used for the member replace option the last time this monitor was started. • REPLACE • ADD Contains the null value for an EVENT monitor.
NUMBER_ROWS	CARD	BIGINT Nullable	The number of rows in the database monitor file. Contains the null value if information is not available.
DATA_SIZE	SIZE	BIGINT Nullable	The total size, in bytes, of the database monitor file. Contains the null value if information is not available.
MONITOR_JOB_FILTER	JOB	VARCHAR(32)	Qualified job name for this monitor. For an EVENT monitor, this is the job that started the monitor. Following the qualified job name is the filter operator that applies to the job name. This is either *EQ or *NE. The special value of *ALL indicates all jobs on the system are monitored. A generic name is allowed for both the job name and the user name.
HOST_VARIABLE	HOSTVAR	VARCHAR(9) Nullable	How host variables are handled in this database monitor. **BASIC** Host variables are written in the QQQ3010 database monitor record. **SECURE** No host variables are captured and no QQQ3010 record is written. **CONDENSED** Host variable values are captured in the QQQ1000 database monitor record in column QQDBCLOB1. No QQQ3010 record is written. Contains the null value for an EVENT monitor.
FORCE_RECORDS	FRCRCD	SMALLINT Nullable	The number of records to be held in the buffer before forcing the records to be written to the file when running with a private monitor. Contains the null value if the system calculates the value or for an EVENT monitor.
RUN_THRESHOLD_FILTER	RUNTHLD	INTEGER Nullable	The filtering threshold, in seconds, based on the estimated run time of SQL statements in this monitor. Contains the null value if a run time threshold is not used for filtering or for an EVENT monitor.
STORAGE_THRESHOLD_FILTER	STGTHLD	INTEGER Nullable	The filtering threshold, in megabytes, based on the estimated temporary storage usage of SQL statements in this monitor. Contains the null value if a temporary threshold is not used for filtering or for an EVENT monitor.
INCLUDE_SYSTEM_SQL	INCSYSSQL	VARCHAR(3)	Monitor includes records for system-generated SQL statements. **YES** Monitor records are generated for both user-specified and system-generated SQL statements. **NO** Monitor records are generated for only user-specified SQL statements. **INI** For a PUBLIC or PRIVATE monitor, records are generated based on the value of the SQL_DBMON_OUTPUT option in the QAQQINI query options.

Table 60. DATABASE_MONITOR_INFO view (continued)

Column Name	System Column Name	Data Type	Description
FILE_FILTER	FTRFILE	VARCHAR(2728) Nullable	A list of up to 10 qualified file references that are used for filtering. Following each file name is the filter operator that applies to the file name. This is either *EQ or *NE. When more than one file is listed, a comma and a single blank separate the entries. Either the file name or the library name can be a generic name. A special value of *ALL for the file name indicates all files in the library. Contains the null value if no database files are used for filtering.
USER_FILTER	FTRUSER	VARCHAR(158) Nullable	A list of up to 10 user profiles that are used for filtering. Following each user profile name is the filter operator that applies to the user profile. This is either *EQ or *NE. When more than one profile is listed, a comma and a single blank separate the entries. A profile name can be a generic name. Contains the null value if the user profile is not used for filtering.
TCPIP_FILTER	FTRINTNETA	VARCHAR(254) Nullable	The TCP/IP address or host name is used for filtering. This is an IPv4, IPv6, or IP host domain name, or the special value of *LOCAL. Contains the null value if the TCP/IP address or host name is not used for filtering or for an EVENT monitor.
LOCAL_PORT_FILTER	FTRLCLPORT	INTEGER Nullable	Filtering is based on the local TCP/IP port number. Monitor records will be created for TCP/IP database server jobs running on behalf of the specified local TCP/IP port. Jobs named QRWTSRVR and QZDASOINIT are examples of these server jobs. The IBM i well defined port numbers are documented here: Port numbers for host servers and server mapper. Contains the null value if the port number is not used for filtering or for an EVENT monitor.
QUERY_GOVERNOR_FILTER	FTRQRYGOVR	VARCHAR(11) Nullable	The query governor is used for filtering. **ALL** Monitor records will be collected when a query governor limit is exceeded. **CONDITIONAL** Monitor records will be conditionally collected when a query governor limit is exceeded. Contains the null value if the query governor is not used for filtering or for an EVENT monitor.
CLIENT_ACCTNG_FILTER	FTRCLTACG	VARCHAR(128) Nullable	The CURRENT CLIENT_ACCTNG special register is used for filtering. Contains the null value if the CURRENT CLIENT_ACCTNG special register is not used for filtering or for an EVENT monitor.
CLIENT_APPLNAME_FILTER	FTRCLTAPP	VARCHAR(128) Nullable	The CURRENT CLIENT_APPLNAME special register is used for filtering. Contains the null value if the CURRENT CLIENT_APPLNAME special register is not used for filtering or for an EVENT monitor.
CLIENT_PROGRAMID_FILTER	FTRCLTPGM	VARCHAR(128) Nullable	The CURRENT CLIENT_PROGRAMID special register is used for filtering. Contains the null value if the CURRENT CLIENT_PROGRAMID special register is not used for filtering or for an EVENT monitor.
CLIENT_USERID_FILTER	FTRCLTUSR	VARCHAR(128) Nullable	The CURRENT CLIENT_USERID special register is used for filtering. Contains the null value if the CURRENT CLIENT_USERID special register is not used for filtering or for an EVENT monitor.
CLIENT_WRKSTNNAME_FILTER	FTRCLTWS	VARCHAR(128) Nullable	The CURRENT CLIENT_WRKSTNNAME special register is used for filtering. Contains the null value if the CURRENT CLIENT_WRKSTNNAME special register is not used for filtering or for an EVENT monitor.

Table 60. DATABASE_MONITOR_INFO view (continued)

Column Name	System Column Name	Data Type	Description
SQL_CODE_FILTER	FTRSQLCODE	VARCHAR(7) Nullable	How the SQLCODE result from a statement execution is used for filtering. **NONZERO** Any SQL statement with an SQLCODE value that is non-zero is included in the monitor. **ERROR** Any SQL statement with an SQLCODE that is less than zero is collected in the monitor. **WARNING** Any SQL statement with an SQLCODE that is greater than zero is collected in the monitor. **SQLCODE** Any SQL statement with an SQLCODE that exactly matches the value in the SQLCODE_VALUE column is collected in the monitor. Contains the null value if the SQLCODE for a statement is not used for filtering or for an EVENT monitor.
SQLCODE_VALUE	SQLCODEVAL	INTEGER Nullable	The positive or negative SQLCODE value to use for filtering. Contains the null value if the SQL_CODE_FILTER column contains a value other than SQLCODE.

Examples

Example 1: Get the MONITOR_ID for all the active PUBLIC monitors and the file names associated with the MONITOR_IDs.

```
SELECT MONITOR_ID, MONITOR_LIBRARY, MONITOR_FILE
  FROM QSYS2.DATABASE_MONITOR_INFO
  WHERE MONITOR_STATUS = 'ACTIVE' AND
        MONITOR_TYPE = 'PUBLIC'
```

Example 2: Find the active monitors that have outfiles larger than 1Gig.

```
SELECT MONITOR_LIBRARY, MONITOR_FILE, NUMBER_ROWS, DATA_SIZE
  FROM QSYS2.DATABASE_MONITOR_INFO
  WHERE MONITOR_STATUS = 'ACTIVE' AND
        DATA_SIZE > 1073741824
```

Example 3: Find any active monitors that are filtering based upon a specific SQLCODE (FTRSQLCODE).

```
SELECT MONITOR_ID, MONITOR_LIBRARY, MONITOR_FILE, SQLCODE_VALUE
  FROM QSYS2.DATABASE_MONITOR_INFO
  WHERE MONITOR_STATUS = 'ACTIVE' AND
        SQL_CODE_FILTER = 'SQLCODE'
```

Example 4: Get the MONITOR_ID for a user's SQL plan cache event monitor and use it to end the active event monitor.

```
CALL QSYS2.END_PLAN_CACHE_EVENT_MONITOR (SELECT MONITOR_ID
                                         FROM QSYS2.DATABASE_MONITOR_INFO
                                         WHERE MONITOR_TYPE = 'EVENT' AND
                                               MONITOR_LIBRARY = 'USERLIB')
```

HARVEST_INDEX_ADVICE procedure

The HARVEST_INDEX_ADVICE procedure generates one or more CREATE INDEX statements in source file members for a specified table based on indexes that have been advised for the table.

```
►► HARVEST_INDEX_ADVICE — ( — schema-name — , — table-name — , —►

►— times_advised — , — mti_used — , — average_estimate — , — output-library — , — output-file —►

►— ) —►◄
```

The schema is SYSTOOLS.

schema-name
A character string containing the system name of the schema containing the table.

table-name
A character string containing the system name of the table.

times-advised
The number of times an index should have been advised before creating a permanent index. Pass a value of 1 to not limit index creation by the number of times advised.

mti-used
The number of times a maintained temporary index (MTI) has been used because a matching permanent index did not exist. Pass a value of 0 to not limit index creation by MTI use.

average-estimate
The average estimated number of seconds needed to execute the query that drove the index advice. Pass a value of 0 to not limit index creation by the average query estimate.

output-library
A character string value containing the name of the library for the output source file.

output-file
A character string value containing the name of the output source file. The file must exist and must be a source physical file.

For each potential index meeting the specified criteria, a CREATE INDEX statement to create the permanent index will be generated in a member in the source file provided to this procedure. A radix index will be named `name_RADIX_INDEX_n`. An EVI index will be named `name_EVI_INDEX_n`. The *name* represents the table name and *n* is a unique number. The row containing this advised index is removed from the QSYS2.SYSIXADV table.

Related reference

SYSTOOLS
SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

REMOVE_INDEXES procedure

The REMOVE_INDEXES procedure drops any indexes meeting the specified criteria.

```
►► REMOVE_INDEXES — ( — schema-name — , — times_used — , — index-age — ) —►◄
```

The schema is SYSTOOLS.

schema-name
A character string containing the system name of the schema containing the indexes to be evaluated. If the NULL value is passed, the entire database is processed.

times_used
A big integer value indicating the number of times an index has been used.

index-age
A character string containing an SQL labeled duration, such as '2 MONTHS'.

The procedure will evaluate all indexes for the specified *schema-name* value and drop any index that does not meet the *times-used* and *index-age* threshold. If the number of times the index has been used by a query and used for statistics is less than the *times-used* value, the index is considered under utilized and is a candidate to be dropped. An index that has existed at least the length of time indicated by *index-age* is also a candidate to be dropped. Any index that meets both criteria is dropped.

Only index names that have names like `name_RADIX_INDEX_x` or `name_EVI_INDEX_x` will be considered by this procedure.

Examples

- Remove any index in MYLIB that is older than a month that has never been used.

```
CALL SYSTOOLS.REMOVE_INDEXES('MYLIB', 1, '1 MONTH')
```

- Remove all indexes from all schemas on the system that have existed for at least two weeks and haven't been used at least 100 times.

```
CALL SYSTOOLS.REMOVE_INDEXES(NULL, 100, '14 DAYS')
```

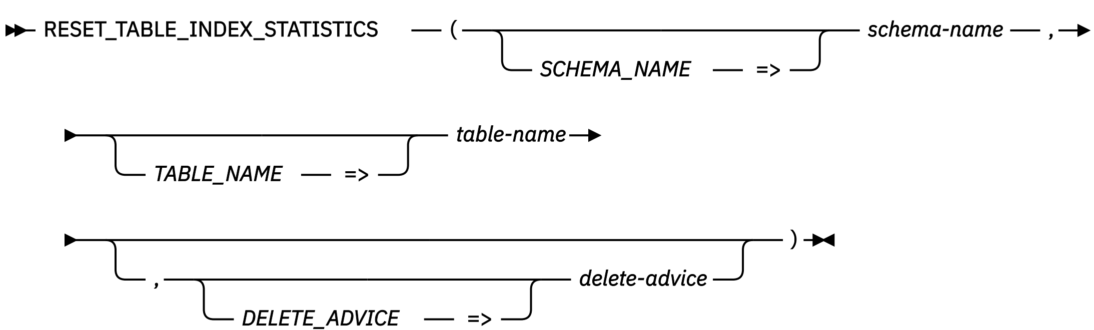
Related reference

SYSTOOLS

SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

RESET_TABLE_INDEX_STATISTICS procedure

The RESET_TABLE_INDEX_STATISTICS procedure clears usage statistics for indexes defined over a table or tables and optionally deletes rows from the index advice tracking table.



The schema is QSYS2.

This procedure will zero the QUERY_USE_COUNT and QUERY_STATISTICS_COUNT usage statistics for all indexes over the specified tables. These counts can also be reset using the Change Object Description (CHGOBJD) CL command, but the command requires an exclusive lock.

schema-name
: A character string expression for the name of the schema or schemas to use. The name is case-sensitive and must not be delimited. Wildcard characters (_ and %) are allowed in the string following the rules for the SQL LIKE predicate.

table-name
: A character string expression for the name of the table or tables to use. The name is case-sensitive and must not be delimited. Wildcard characters (_ and %) are allowed in the string following the rules for the SQL LIKE predicate.

delete-advice
: A character string expression that indicates whether this procedure should remove rows from the index advice tracking table.

NO
: Index advice for the table is not affected. This is the default.

YES
: This procedure will delete rows from the index advice tracking table (QSYS2/SYSIXADV) that correspond to *schema-name* and *table-name*.

Authorization: The counts will only be reset when the caller has *OBJMGT and *OBJOPR authority on the table. For each index found over the table, *OBJOPR is required. If the user does not have the required authority to the table, the object is skipped and no warning is returned. If the user does not have the required authority to the index, the object is skipped and an SQL warning is returned. To delete index advice, the DELETE privilege is required on QSYS2/SYSIXADV. Index advice is only deleted when the caller has the required authority to the table and index.

The procedure writes information related to every index processed into an SQL global temporary table. The fields LAST_QUERY_USE, LAST_STATISTICS_USE, LAST_USE_DATE, and NUMBER_DAYS_USED are not affected. The following query will display the results of the last call to the procedure:

SELECT * FROM SESSION.SQL_Indexes_Reset

The table that is created contains the following columns:

Table 61. SQL_Indexes_Reset result table

Column Name	System Column Name	Data Type	Description
TABLE_SCHEMA	DBNAME	VARCHAR(128)	Schema name of table.
TABLE_NAME	NAME	VARCHAR(128)	Name of table.
TABLE_PARTITION	TABLE00001	VARCHAR(128)	Name of the table partition or member.
PARTITION_TYPE	PARTI00001	CHAR(1)	The type of table partitioning.
PARITION_NUMBER	PARTI00002	INTEGER	The partition number of this partition.
NUMBER_DISTRIBUTED_PARTITIONS	NUMBE00001	INTEGER	If the table is a distributed table, contains the total number of partitions.
INDEX_SCHEMA	INDEX00001	VARCHAR(128)	Schema name of index.
INDEX_NAME	INDEX_NAME	VARCHAR(128)	Name of index.
INDEX_MEMBER	INDEX00002	VARCHAR(128)	Partition or member name of index.
INDEX_TYPE	INDEX_TYPE	CHAR(11)	Type of index.
LAST_QUERY_USE	LAST_00002	TIMESTAMP	The timestamp of the last time the SQL index was used in a query since the last time the usage statistics were reset.
LAST_STATISTICS_USE	LAST_00003	TIMESTAMP	The timestamp of the last time the SQL index was used by the optimizer for statistics since the last time the usage statistics were reset.
QUERY_USE_COUNT	QUERY00001	BIGINT	The number of times the SQL index was used in a query since the last time the usage statistics were reset.
QUERY_STATISTICS_COUNT	QUERY00002	BIGINT	The number of times the SQL index was used by the optimizer for statistics since the last time the usage statistics were reset.
SYSTEM_TABLE_SCHEMA	SYS_DNAME	CHAR(10)	System table schema name.
SYSTEM_TABLE_NAME	SYS_TNAME	CHAR(10)	System table name.
SYSTEM_TABLE_MEMBER	SYSTE00001	CHAR(10)	System member name.

Examples

- Zero the statistics for all indexes over table TOYSTORE.SALES

```
CALL qsys2.Reset_Table_Index_Statistics ('TOYSTORE', 'SALES')
```

- Zero the statistics for all indexes over any table in schema TOYSTORE whose name starts with the letter S.

```
CALL qsys2.Reset_Table_Index_Statistics ('TOYSTORE', 'S%')
```

Utility Services

These procedures provide interfaces to monitor and work with SQL in jobs on the current system or to compare constraint and routine information across systems.

CANCEL_SQL procedure

The CANCEL_SQL procedure requests cancellation of an SQL statement for the specified job.

```
►►─ CANCEL_SQL ── ( ── job-name ── ) ─►◄
```

The schema is QSYS2.

job-name
A character string containing the qualified job name to be cancelled. It must be in upper case.

The CANCEL_SQL() procedure provides an alternative to end job immediate. It supports all application and interactive SQL environments.

When an SQL cancel is requested, an asynchronous request is sent to the job identified by *job-name*. If the job is processing an interruptible, long-running machine operation, analysis is done within the job to determine whether it is safe to cancel the statement. When it is determined to be safe to cancel the statement, an SQL0952 escape message is sent, causing the statement to terminate.

If it isn't safe to end the SQL statement, or if there is no active SQL statement, the request to cancel is ignored. The caller of the cancel procedure will observe a successful return which only indicates that the caller had the necessary authority to request a cancel and that the target job exists. The caller of the CANCEL_SQL() procedure has no programmatic means of determining that the cancel request resulted in a cancelled SQL statement.

If the cancel request occurs during the act of committing or rolling back a commitment-control transaction, the request is ignored.

Authorization: The CANCEL_SQL procedure requires that the authorization ID associated with the statement has *JOBCTL special authority.

Errors: The procedure will fail with a SQL0443 if the target job is not found. The procedure will fail with SQL0443 and SQL0552 if the caller does not have *JOBCTL user special authority.

Commitment control: When the target application is running without commitment control (COMMIT = *NONE), the cancelled SQL statement will terminate without rolling back the partial results of the statement. If the cancelled statement is a query, the query ends. However, if the cancelled statement was a long-running INSERT, UPDATE, or DELETE SQL statement, the changes made prior to cancellation remain intact.

If the target application is using transaction management, the SQL statement is running under a transaction savepoint level. When a long running INSERT, UPDATE, or DELETE SQL statement is cancelled, the changes made prior to cancellation are rolled back.

In both cases, the application receives control back with an indication that the SQL statement failed. It is up to the application to determine the next action.

Example

Safely cancel a job running an SQL statement.

```
CALL QSYS2.CANCEL_SQL('483456/QUSER/QZDASOINIT')
```

CHECK_SYSCST procedure

The CHECK_SYSCST procedure compares entries in the QSYS2.SYSCONSTRAINTS table between two systems.

```
>>- CHECK_SYSCST -- ( -- remote-rdb-name -- , -- schema-name --+---------------------------+-- ) -><
                                                               '-- , -- avoid-result-set --'
```

The schema is SYSTOOLS.

remote-rdb-name
A character string containing the name of the remote database.

schema-name
A character string containing the name of the schema to compare.

avoid-result-set
An integer value that indicates whether a result set should be returned. The default is 0.

1
No result set is returned.

0
Result set is returned.

This procedure will return a result set to the caller. If no result set is requested, the differences are logged to the SESSION.SYSCSTDIFF table.

The result set that is returned or the table that is created contains the following columns:

Table 62. SYSCSTDIFF result set

Column Name	System Column Name	Data Type	Description
SERVER_NAME	SRVRNAME	VARCHAR(18)	Name of server where the request was run.
CONSTRAINT_SCHEMA	CDNAME	VARCHAR(128)	Name of the schema containing the constraint.
CONSTRAINT_NAME	RELNAME	VARCHAR(128)	Name of the constraint.
CONSTRAINT_TYPE	TYPE	VARCHAR(11)	Type of constraint.
TABLE_SCHEMA	TDBNAME	VARCHAR(128)	Name of schema containing the table.
TABLE_NAME	TBNAME	VARCHAR(128)	Name of the table which the constraint is created over.
SYSTEM_TABLE_SCHEMA	SYS_DNAME	CHAR(10)	System name of schema containing the table.
SYSTEM_TABLE_NAME	SYS_TNAME	CHAR(10)	System name of the table which the constraint is created over.
CONSTRAINT_KEYS	COLCOUNT	SMALLINT	Specifies the number of key columns if this is a UNIQUE, PRIMARY KEY, or FOREIGN KEY constraint.
CONSTRAINT_STATE	CST_STATE	VARCHAR(11)	Indicates whether the constraint is established or defined.
ENABLED	ENABLED	VARCHAR(3)	Indicates whether the constraint is enabled or disabled.
CHECK_PENDING	CHECK00001	VARCHAR(3)	Indicates whether the constraint is in check pending state.

Example

Find all the differences in constraint settings between the current system and LP01UT18 for the CORPDB_EX schema:

```
CALL SYSTOOLS.CHECK_SYSCST('LP01UT18', 'CORPDB_EX')
```

Related reference
SYSTOOLS

SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

CHECK_SYSROUTINE procedure

The CHECK_SYSROUTINE procedure compares entries in the QSYS2.SYSROUTINES table between two systems.

```
>>- CHECK_SYSROUTINE -- ( -- remote-rdb-name -- , -- schema-name --+--------------------------+-->
                                                                   '-- , -- avoid-result-set -'

>-- ) -><
```

The schema is SYSTOOLS.

remote-rdb-name
A character string containing the name of the remote database.

schema-name
A character string containing the name of the schema to compare.

avoid-result-set
An integer value that indicates whether a result set should be returned. The default is 0.

1
No result set is returned.

0
Result set is returned.

This procedure will return a result set to the caller. If no result set is requested, the differences are logged to the SESSION.SYSRTNDIFF table.

The result set that is returned or the table that is created contains the following columns:

Table 63. SYSRTNDIFF result set

Column Name	System Column Name	Data Type	Description
SERVER_NAME	SRVRNAME	VARCHAR(18)	Name of server where the request was run.
ROUTINE_CREATED	RTNCREATE	TIMESTAMP	The timestamp when the routine was created.
ROUTINE_DEFINER	DEFINER	VARCHAR(128)	Name of the user that defined the routine.
LAST_ALTERED	ALTEREDTS	TIMESTAMP	Timestamp when routine was last altered.
SPECIFIC_SCHEMA	SPECSCHEMA	VARCHAR(128)	Schema name of the routine instance.
SPECIFIC_NAME	SPECNAME	VARCHAR(128)	Specific name of the routine instance.
ROUTINE_SCHEMA	RTNSCHEMA	VARCHAR(128)	Name of the schema that contains the routine.
ROUTINE_NAME	RTNNAME	VARCHAR(128)	Name of the routine.
ROUTINE_TYPE	RTNTYPE	VARCHAR(9)	Type of the routine.
ROUTINE_BODY	BODY	VARCHAR(8)	Type of the routine body.
EXTERNAL_NAME	EXTNAME	VARCHAR(279)	External program name for routine.
IN_PARMS	IN_PARMS	SMALLINT	Identifies the number of input parameters.
OUT_PARMS	OUT_PARMS	SMALLINT	Identifies the number of output parameters.
INOUT_PARMS	INOUT_PARM	SMALLINT	Identifies the number of input/output parameters.
SQL_DATA_ACCESS	DATAACCESS	VARCHAR(8)	Identifies whether a routine contains SQL and whether it reads or modifies data.
PARM_SIGNATURE	SIGNATURE	VARCHAR(2048)	The signature of the routine.

Example

Compare the current system to a remote system to find which routines differ, when they were created, and who created them.

```
CALL SYSTOOLS.CHECK_SYSROUTINE('LP01UT18', 'CORPDB_EX')
```

Related reference

SYSTOOLS

SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

DUMP_SQL_CURSORS procedure

The DUMP_SQL_CURSORS procedure lists the open cursors for a job.

```
►►─ DUMP_SQL_CURSORS ── ( ─►

►── job-name ── , ── library-name ── , ── table-name ── , ── output-option ── ) ─►◄
```

The schema is QSYS2.

job-name
A character string containing a qualified job name or a value of '*' to indicate the current job

library-name
A character string containing a system library name for the procedure output. An empty string is allowed.

table-name
A character string containing a system table name for the procedure output. An empty string is allowed.

output-option
An integer value that indicates how to return the output.

1
Ignore *library-name* and *table-name* parameters and return a result set.

2
Ignore *library-name* and *table-name* parameters and place the results in table QTEMP/SQL_CURSORS.

3
Place the results in table *table-name* in library *library-name*. If the table doesn't exist, it will be created. If the table already exists, the results will be appended to the existing table.

4
Place the results in table *table-name* in library *library-name*. If the table doesn't exist, it will not be created.

The result set that is returned or the table that is created contains the following columns:

Table 64. DUMP_SQL_CURSORS result table

Column Name	System Column Name	Data Type	Description
SQL_IDENTITY	SQL_I00001	INTEGER	Unique identifier for the row.
DUMPTIME	DUMPTIME	TIMESTAMP	Timestamp when row was inserted.
DUMP_BY_USER	DUMPUSER	VARCHAR(18)	User ID used to insert row.
CURSOR_NAME	CSRNAME	VARCHAR(128)	Name of the cursor.
PSEUDO_CLOSED	PSEUDO	VARCHAR(3)	Pseudo close state of the cursor. Valid values are: **YES** Cursor is currently pseudo closed. **NO** Cursor is currently opened.
STATEMENT_NAME	STMTNAME	VARCHAR(128)	Name of the statement corresponding to the cursor

Table 64. DUMP_SQL_CURSORS result table (continued)

Column Name	System Column Name	Data Type	Description
OBJECT_NAME	OBJNAME	CHAR(10)	Object containing the current SQL statement. Blank if current SQL statement is not in a program, service program, or package.
OBJECT_LIBRARY	OBJLIB	CHAR(10)	Library for object containing the current SQL statement. Blank if current SQL statement is not in a program, service program, or package.
OBJECT_TYPE	OBJTYPE	CHAR(10)	Type of object containing the current SQL statement. Blank if current SQL statement is not in a program, service program, or package.
JOBNAME	JOBNAME	CHAR(28)	System job name for the cursor. Contains * if current job was specified for *job-name* argument.

Example

Populate QGPL/SQLCSR1 table with open SQL cursors for the current job.

```
CALL QSYS2.DUMP_SQL_CURSORS('*', 'QGPL', 'SQLCSR1', 3);
```

FIND_AND_CANCEL_QSQSRVR_SQL procedure

The FIND_AND_CANCEL_QSQSRVR_SQL procedure finds a set of jobs with SQL activity and safely cancels them.

▶▶─ FIND_AND_CANCEL_QSQSRVR_SQL ── (── *job-name* ──) ─▶◀

The schema is QSYS2.

job-name
A character string containing a qualified job name.

The FIND_AND_CANCEL_QSQSRVR_SQL procedure uses the FIND_QSQSRVR_JOBS and CANCEL_SQL procedures to determine the set of jobs that have SQL activity for the provided *job-name*. Each of these jobs is made a target of an SQL cancel request.

Example

Cancel all the QSQSRVR jobs used by a specific job.

```
CALL QSYS2.FIND_AND_CANCEL_QSQSRVR_SQL('564321/APPUSER/APPJOBNAME')
```

FIND_QSQSRVR_JOBS procedure

The FIND_QSQSRVR_JOBS procedure returns information about a QSQSRVR job.

▶▶─ FIND_QSQSRVR_JOBS ── (── *job-name* ──) ─▶◀

The schema is QSYS2.

job-name
A character string containing a qualified job name.

If the specified job is active and is set up to use SQL server mode, the procedure determines which QSQSRVR jobs are being used by the application in the form of active SQL server mode connections. The procedure collects and returns work management, performance, and SQL information. It returns two SQL result sets, one containing summary information and one containing detailed SQL server mode job information.

Authorization: To invoke FIND_QSQSRVR_JOBS you need *JOBCTL special authority, QIBM_DB_SQLADM function usage, or QIBM_DB_SYSMON function usage.

The results of the procedure call are saved in two temporary tables, QTEMP.QSQSRVR_SUMMARY and QTEMP.QSQSRVR_DETAIL. When called from within IBM i Navigator Run SQL Scripts, two results sets are displayed. When called from other interfaces, you need to query the temporary tables to see the data.

The result sets that are returned or the tables that are created contain the following columns:

Table 65. FIND_QSQSRVR_JOBS result set 1

Column Name	System Column Name	Data Type	Description
SQL_IDENTITY	SQL_I00001	INTEGER	Unique identifier for this row.
NUMBER_OF_ACTIVE_JOBS	NUMJOBS	INTEGER	Number of QSQSRVR jobs active for this job.
SERVER_MODE_JOB	SRVRJOB	CHAR(28)	The fully qualified QSQSRVR job name for an active SQL Server Mode connection established by *job-name*.
SERVER_MODE_CONNECTING_JOB	CONNJOB	CHAR(28)	The fully qualified job name of the application job. This value matches what was input for *job_name*.
TOTAL_PROCESSING_TIME	TOTALCPU	BIGINT	The total amount of CPU time (in milliseconds) that has been used by all server jobs.
TEMP_MEG_STORAGE	TEMPMSTG	INTEGER	The total amount of auxiliary storage (in megabytes) that is currently allocated to all server jobs.
PAGE_FAULTS	FAULTS	BIGINT	The total number of times an active program referenced an address that was not in main storage for all server jobs.
IO_REQUESTS	IOREQS	BIGINT	The total number of auxiliary I/O requests performed by the job across all routing steps for all server jobs. This includes both database and non-database paging.

Table 66. FIND_QSQSRVR_JOBS result set 2

Column Name	System Column Name	Data Type	Description
SQL_IDENTITY	SQL_I00001	INTEGER	Unique identifier for this row.
JOB_NAME	JOBNAME	CHAR(10)	Job name.
USER_NAME	USERNAME	CHAR(10)	User ID for the job.
JOB_NUMBER	JOBNUM	CHAR(6)	Job number.
JOB_INTERNAL_IDENTIFIER	JOBID	CHAR(16)	Internal identifer assigned to job.
CURRENT_USERNAME	CURRUSER	CHAR(10)	The user profile that the thread is currently running under.
SUBSYSTEM_DESCRIPTION_NAME	SBSNAME	CHAR(10)	Name of subsystem where job is running.
RUN_PRIORITY	PRIORITY	INTEGER	The highest run priority allowed for any thread within this job.
SYSTEM_POOL_IDENTIFIER	POOLID	INTEGER	The identifier of the system-related pool from which the job's main storage is allocated.
TOTAL_PROCESSING_TIME	TOTALCPU	BIGINT	The amount of CPU time (in milliseconds) that has been currently used by this job.
PAGE_FAULTS	FAULTS	BIGINT	The number of times an active program referenced an address that was not in main storage during the current routing step of the specified job.
IO_REQUESTS	IOREQS	BIGINT	The number of auxiliary I/O requests performed by the job across all routing steps. This includes both database and non-database paging.
MEMORY_POOL_NAME	POOLNAME	CHAR(10)	The name of the memory pool in which the job started running.
TEMP_MEG_STORAGE	TEMPMSTG	INTEGER	The amount of auxiliary storage (in megabytes) that is currently allocated to this job.
TIME_SLICE	TSLICE	INTEGER	The maximum amount of processor time (in milliseconds) given to each thread in this job before other threads in this job and in other jobs are given the opportunity to run.
DEFAULT_WAIT	DFTWAIT	INTEGER	The default maximum time (in seconds) that a thread in the job waits for a system instruction to acquire a resource.

Table 66. FIND_QSQSRVR_JOBS result set 2 (continued)

Column Name	System Column Name	Data Type	Description
SQL_APPLICATION_LIBRARY	SQLLIB	CHAR(10)	The library name for the SQL statement object.
SQL_APPLICATION_PROGRAM	SQLPGM	CHAR(10)	The program, service program, or package name of the object which contains the last SQL statement executed in the job.
SQL_APPLICATION_TYPE	APPTYPE	CHAR(10)	The object type.
SERVER_MODE_CONNECTING_JOB	CONNJOB	CHAR(28)	The qualified job name of the job which established the SQL Server Mode connection.
SERVER_MODE_CONNECTED_THREAD	CONNTHD	CHAR(10)	The thread identifier of the last thread to use this connection.
STATUS_OF_CURRENT_SQL_STMT	STMTSTAT	CHAR(10)	Status of the SQL statement. Values are ACTIVE or COMPLETED.
SQL_STATEMENT	SQLSTMT	VARCHAR(1000)	First 1000 characters of the SQL statement.

GENERATE_SQL procedure

The GENERATE_SQL procedure generates the SQL data definition language statements required to recreate a database object. The results are returned in the specified database source file member or as a result set.

The database source file member will contain the generated SQL statements. If the output source file is QTEMP/Q_GENSQL with a member name of Q_GENSQL, the source file is returned as a result set as well.

```
>>- GENERATE_SQL -- ( --+-----------------------------+-- database-object-name -- , -->
                        '-- DATABASE_OBJECT_NAME -- => -'

>--+-------------------------------------+-- database-object-library-name -- , -->
   '-- DATABASE_OBJECT_LIBRARY_NAME -- => -'

>--+-----------------------------+-- database-object-type -->
   '-- DATABASE_OBJECT_TYPE -- => -'

>--+------------------------------------------------------------------+-->
   '-- , --+----------------------------------+-- database-source-file-name --'
           '-- DATABASE_SOURCE_FILE_NAME -- => -'

>--+----------------------------------------------------------------------------------+-->
   '-- , --+------------------------------------------+-- database-source-file-library-name --'
           '-- DATABASE_SOURCE_FILE_LIBRARY_NAME -- => -'

>--+----------------------------------------------------------------------+-->
   '-- , --+------------------------------------+-- database-source-file-member --'
           '-- DATABASE_SOURCE_FILE_MEMBER -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- severity-level --'
           '-- SEVERITY_LEVEL -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- replace-option --'
           '-- REPLACE_OPTION -- => -'

>--+------------------------------------------------------------------------+-->
   '-- , --+-------------------------------------+-- statement-formatting-option --'
           '-- STATEMENT_FORMATTING_OPTION -- => -'

>--+-----------------------------------------------+-->
   '-- , --+---------------------+-- date-format --'
           '-- DATE_FORMAT -- => -'

>--+-----------------------------------------------------+-->
   '-- , --+------------------------+-- date-separator --'
           '-- DATE_SEPARATOR -- => -'

>--+-----------------------------------------------+-->
   '-- , --+---------------------+-- time-format --'
           '-- TIME_FORMAT -- => -'

>--+-----------------------------------------------------+-->
   '-- , --+------------------------+-- time-separator --'
           '-- TIME_SEPARATOR -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- naming-option --'
           '-- NAMING_OPTION -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- decimal-point --'
           '-- DECIMAL_POINT -- => -'

>--+-----------------------------------------------------------+-->
   '-- , --+---------------------------+-- standards-option --'
           '-- STANDARDS_OPTION -- => -'

>--+-----------------------------------------------+-->
   '-- , --+---------------------+-- drop-option --'
           '-- DROP_OPTION -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- message-level --'
           '-- MESSAGE_LEVEL -- => -'

>--+-------------------------------------------------------+-->
   '-- , --+-------------------------+-- comment-option --'
           '-- COMMENT_OPTION -- => -'

>--+-----------------------------------------------+-->
   '-- , --+---------------------+-- label-option --'
           '-- LABEL_OPTION -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- header-option --'
           '-- HEADER_OPTION -- => -'

>--+---------------------------------------------------+-->
   '-- , --+-----------------------+-- trigger-option --'
           '-- TRIGGER_OPTION -- => -'

>--+-----------------------------------------------------------+-->
   '-- , --+----------------------------+-- constraint-option --'
           '-- CONSTRAINT_OPTION -- => -'

>--+-----------------------------------------------------------------+-->
   '-- , --+------------------------------+-- system-name-option --'
           '-- SYSTEM_NAME_OPTION -- => -'

>--+-----------------------------------------------------------+-->
   '-- , --+----------------------------+-- privileges-option --'
           '-- PRIVILEGES_OPTION -- => -'

>--+-----------------------------------------------+-->
   '-- , --+---------------------+-- ccsid-option --'
           '-- CCSID_OPTION -- => -'

>--+-----------------------------------------------------------------------+-->
   '-- , --+-----------------------------------+-- create-or-replace-option --'
           '-- CREATE_OR_REPLACE_OPTION -- => -'

>--+-----------------------------------------------------------+-->
   '-- , --+----------------------------+-- obfuscate-option --'
           '-- OBFUSCATE_OPTION -- => -'

>--+-----------------------------------------------------------------------------------------------+-->
   '-- , --+--------------------------------------------------------+-- activate-access-control-option --'
           '-- ACTIVATE_ROW_AND_COLUMN_ACCESS_CONTROL_OPTION -- => -'

>--+--------------------------------------------------------------------------+-->
   '-- , --+------------------------------------+-- mask-and-permission-option --'
           '-- MASK_AND_PERMISSION_OPTION -- => -'

>--+--------------------------------------------------------------------+-->
   '-- , --+--------------------------------+-- qualified-name-option --'
           '-- QUALIFIED_NAME_OPTION -- => -'

>--+----------------------------------------------------------------------+-->
   '-- , --+---------------------------------+-- additional-index-option --'
           '-- ADDITIONAL_INDEX_OPTION -- => -'

>--+----------------------------------------------------------------------------+-->
   '-- , --+------------------------------------+-- index-instead-of-view-option --'
           '-- INDEX_INSTEAD_OF_VIEW_OPTION -- => -'

>-- ) -><
```

The schema is QSYS2.

database-object-name
: A character or graphic string expression that identifies the name of the database object for which DDL will be generated. Either the SQL name or the system name may be specified. The name is case

sensitive. Delimiters must not be specified. For example, a file with a name of "abc" must be specified as `abc`. A file with a name of ABC must be specified in upper case. If the object type is a FUNCTION or PROCEDURE, this name must be the specific name of the function or procedure. If TABLE or VIEW is specified for the object type, the object name may identify an alias. In this case, the object that the alias points to will be generated. A CREATE ALIAS statement will be generated only if ALIAS is specified for the object type.

A '%' wildcard character can be used to select multiple objects of the same type. For example, a name of `'TSTV%'` will process all objects of *database-object-type* that start with the characters `'TSTV'`

database-object-library-name
A character or graphic string expression that identifies the name of the library containing the object for which DDL will be generated. Either the SQL name or the system name may be specified. The name is case sensitive. Delimiters must not be specified. This name is ignored if the specified object type is SCHEMA. A '%' wildcard character can be used to select multiple libraries.

database-object-type
A character or graphic string expression that identifies the type of the database object or object attribute for which DDL is generated. You can use these special values for the object type:

ALIAS
The object is an SQL alias.

CONSTRAINT
The object attribute is a constraint.

FUNCTION
The object is an SQL function.

INDEX
The object is an SQL index.

MASK
The object is an SQL column mask.

PERMISSION
The object is an SQL row permission.

PROCEDURE
The object is an SQL procedure.

SCHEMA
The object is an SQL schema.

SEQUENCE
The object is an SQL sequence.

TABLE
The object is an SQL table or physical file.

TRIGGER
The object attribute is a trigger.

TYPE
The object is an SQL type.

VARIABLE
The object is an SQL global variable.

VIEW
The object is an SQL view or logical file.

XSR
The object is an XML schema repository object.

database-source-file-name
A character or graphic string expression that identifies the name of the source file that contains the SQL statements generated by the procedure. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. For

example, a file with a name of "abc" must be specified with the surrounding quotes. A file with a name of ABC must be specified in upper case.

The record length of the specified source file must be greater than or equal to 92.

If *database-source-file-name* is not specified, `Q_GENSQL` will be used.

database-source-file-library

A character or graphic string expression that identifies the name of the library containing the source file that contains the SQL statements generated by the procedure. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified.

If *database-source-file-library* is not specified, `QTEMP` will be used.

database-source-file-member

A character or graphic string expression that identifies the name of the source file member that contains the SQL statements generated by the procedure. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. You can use these special values for the member name:

***FIRST**

The first database physical file member found.

***LAST**

The last database physical file member found.

If values are provided for *database-source-file-library-name*, *database-source-file-name*, and *database-source-file-member* the object must exist.

If *database-source-file-member* is not specified, `Q_GENSQL` will be used.

severity-level

The severity level at which the operation fails. If errors occur that have a severity level greater than this value, the operation ends. The valid values are in the range 0 through 39 inclusive. Any severity 40 error will cause the procedure to fail.

If *severity-level* is not specified, 39 will be used.

replace-option

The replace option for the database source file member. The valid values are:

0

The resulting SQL statements are appended to the end of the database source file member.

1

The database source file member is cleared prior to adding the resulting SQL statements. If this option is chosen, the member may be cleared even if an error is returned from the procedure. If multiple objects are being generated, the member is cleared for each object so only the last generated statement will remain in the member.

If *replace-option* is not specified, 1 will be used.

statement-formatting-option

The formatting option used in the generated SQL statements. The valid values are:

0

No additional formatting characters are added to the generated SQL statements.

1

Additional end-of-line characters and tab characters are added to the generated SQL statements.

If *statement-formatting-option* is not specified, 1 will be used.

date-format

The date format used for date constants in a generated SQL CREATE TABLE statement. The date format may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION.

If *date-format* is not specified, ISO will be used.

date-separator
The date separator used for date constants in a generated SQL CREATE TABLE statement. The date separator may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION statement.

If *date-separator* is not specified, - will be used.

time-format
The format used for time constants in a generated SQL CREATE TABLE statement. The time format may not apply to time constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION statement.

If *time-format* is not specified, ISO will be used.

time-separator
The time separator used for time constants in a generated SQL CREATE TABLE statement. The time separator may not apply to time constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION statement.

If *time-separator* is not specified, . will be used.

naming-option
The naming convention used for qualified names in the generated SQL statements. The valid values are:

SQL
schema.table syntax

SYS
library/file syntax

If *naming-option* is not specified, SQL will be used.

decimal-point
The decimal point used for numeric constants. The valid values are:

.
Period separator

,
Comma separator

If *decimal-point* is not specified, . will be used.

standards-option
The standards option specifies whether the generated SQL statements should contain Db2 for i extensions or whether the statements should conform to the Db2 family SQL or to the ANS and ISO SQL standards. The valid values are:

0
Db2 for i extensions may be generated in SQL statements.

1
The generated SQL statements must conform to SQL statements common to the Db2 family.

2
The generated SQL statements must conform to the ANSI and ISO SQL standards.

If *standards-option* is not specified, 0 will be used.

drop-option
The drop option specifies whether DROP (or ALTER) SQL statements should be generated prior to the CREATE statement to drop the specified object. The valid values are:

0
: DROP statements should not be generated.

1
: DROP statements should be generated.

If *drop-option* is not specified, `0` will be used.

message-level

The severity level at which the messages are generated. If errors occur that have a severity level greater than this value, a message is generated in the output. The valid values are in the range 0 through 39 inclusive. The message level must be less than or equal to the severity level.

If *message-level* is not specified, `0` will be used.

comment-option

The comment option specifies whether COMMENT ON SQL statements should be generated if a comment exists on the specified database object. If comments are not supported by the specified database object, the comment option is ignored. The valid values are:

0
: COMMENT ON SQL statements should not be generated.

1
: COMMENT ON SQL statements should be generated. If the specified database object type is a table or view, COMMENT ON SQL statements will also be generated for columns of the table or view.

If *comment-option* is not specified, 1 will be used.

label-option

The label option specifies whether LABEL ON SQL statements should be generated if a label exists on the specified database object. If labels are not supported by the specified database object, the label option is ignored. The valid values are:

0
: LABEL ON SQL statements should not be generated.

1
: LABEL ON SQL statements should be generated. If the specified database object type is a table or view, LABEL ON SQL statements will also be generated for columns of the table or view.

If *label-option* is not specified, 1 will be used.

header-option

The header option specifies whether a header should be generated prior to the CREATE statement. The header consists of comments that describe the version, date and time, the relational database, and some of the options used to generate the SQL statements. The valid values are:

0
: A header should not be generated.

1
: A header should be generated.

If *header-option* is not specified, 1 will be used.

trigger-option

The trigger option specifies whether triggers should be generated when the object type is a TABLE or VIEW. The valid values are:

0
: Triggers should not be generated.

1
: Triggers should be generated.

If *trigger-option* is not specified, 1 will be used.

constraint-option
The constraint option specifies whether constraints should be generated when the object type is a TABLE. The valid values are:

0
Constraints should not be generated.

1
Constraints should be generated using ALTER TABLE statements.

2
Constraints should be generated as part of the CREATE TABLE statement.

If *constraint-option* is not specified, 1 will be used.

system-name-option
The system name option specifies whether a FOR SYSTEM NAME clause should be generated for the system name when it is different from the SQL name and the object type is an INDEX, TABLE, VIEW, or VARIABLE. The valid values are:

0
A FOR SYSTEM NAME clause should not be generated.

1
A FOR SYSTEM NAME clause should be generated.

If *system-name-option* is not specified, 1 will be used.

privileges-option
The privileges option specifies whether GRANT SQL statements should be generated on the specified database object. If privileges are not supported by the specified database object, the privileges option is ignored. The valid values are:

0
GRANT SQL statements should not be generated.

1
GRANT SQL statements should be generated.

If *privileges-option* is not specified, 1 will be used.

ccsid-option
The CCSID option specifies whether the CCSID attribute should be generated for column definitions when the object type is a TABLE. The valid values are:

0
CCSID attribute should not be generated.

1
CCSID attribute should be generated.

If *ccsid-option* is not specified, 1 will be used.

create-or-replace-option
The create or replace option specifies whether CREATE OR REPLACE should be generated for the specified database object on the CREATE statement. This option is ignored if the specified database object does not support CREATE OR REPLACE. The valid values are:

0
CREATE OR REPLACE should not be generated.

1
CREATE OR REPLACE should be generated.

If *create-or-replace-option* is not specified, `0` will be used.

obfuscate-option
The obfuscate option specifies whether an obfuscated SQL statement should be returned for SQL functions, SQL procedures, or SQL triggers that were not created using obfuscated statements. This option is ignored if the standards option is not ‘0’. This option is also ignored if the object is not an SQL

function, procedure, or trigger. This option is ignored if the object is already obfuscated. Setting Obfuscate option = 0 cannot be used as a means of obtaining the unobfuscated SQL statement for an obfuscated object. The valid values are:

0
: An obfuscated statement should not be generated.

1
: An obfuscated statement should be generated for SQL functions, SQL procedures, or SQL triggers.

If *obfuscate-option* is not specified, `0` will be used.

activate-access-control-option

The activate row and column access control option specifies whether an ALTER TABLE to activate row and column access control should be generated when the object type is a TABLE. This option is ignored if the standards option is not ‘0’ or '1'. The valid values are:

0
: Activate row and column access control should not be generated.

1
: Activate row and column access control should be generated.

If *activate-access-control-option* is not specified, `1` will be used.

mask-and-permission-option

The mask and permission option specifies whether row permissions and column masks should be generated when the object type is a TABLE. This option is ignored if the standards option is not ‘0’ or '1'. The valid values are:

0
: Permissions and masks should not be generated.

1
: Permissions and masks should be generated.

If *mask-and-permission-option* is not specified, `1` will be used.

qualified-name-option

The qualified name option specifies whether qualified or unqualified names should be generated for the specified database object. The valid values are:

0
: Qualified object names should be generated. Unqualified names within the body of SQL routines will remain unqualified.

1
: Unqualified object names should be generated when the a library is found which matches the database object library name. Any SQL object or column reference that is RDB qualified will be generated in its fully qualified form. For example, rdb-name.schema-name.table-name and rdb-name.schema-name.table-name.column-name references will retain their full qualification.

If *qualified-name-option* is not specified, `0` will be used.

additional-index-option

The additional index option specifies whether additional CREATE INDEX statements will be generated for DDS-created keyed physical or logical files. The valid values are:

0
: Additional CREATE INDEX statements will not be generated.

1
: An additional CREATE INDEX statement will be generated that matches the index for a DDS-created keyed physical file. If the physical file has a PRIMARY KEY constraint, a CREATE INDEX statement is not generated.

  An additional CREATE INDEX statement will be generated that matches the index for a DDS-created keyed logical file. If a value of ‘1’ is specified for the index instead of view option, an additional CREATE INDEX statement is not generated. Additional CREATE INDEX statements will also be generated that match the join indexes of a DDS-created join logical file.

If *additional-index-option* is not specified, `0` will be used.

index-instead-of-view-option

The Index instead of view option specifies whether a CREATE INDEX or CREATE VIEW statement will be generated for a DDS-created keyed logical file. The valid values are:

0

A CREATE VIEW statement will be generated.

1

A CREATE INDEX statement will be generated that matches the index for a DDS-created keyed logical file.

If *index-instead-of-view-option* is not specified, `0` will be used.

Examples

- Generate DDL for all tables in a schema and return the source as a result set.

```
CALL QSYS2.GENERATE_SQL('%', 'SAMPLE_CORPDB', 'TABLE', REPLACE_OPTION => '0');
```

- Generate DDL for all indexes starting with 'X' within the SAMPLE_CORPDB schema, place the output in a file named DDLSOURCE/GENFILE member INDEXSRC.

```
CALL QSYS2.GENERATE_SQL('X%', 'SAMPLE_CORPDB', 'INDEX',
                        'GENFILE', 'DDLSOURCE', 'INDEXSRC',
                        REPLACE_OPTION => '0');
```

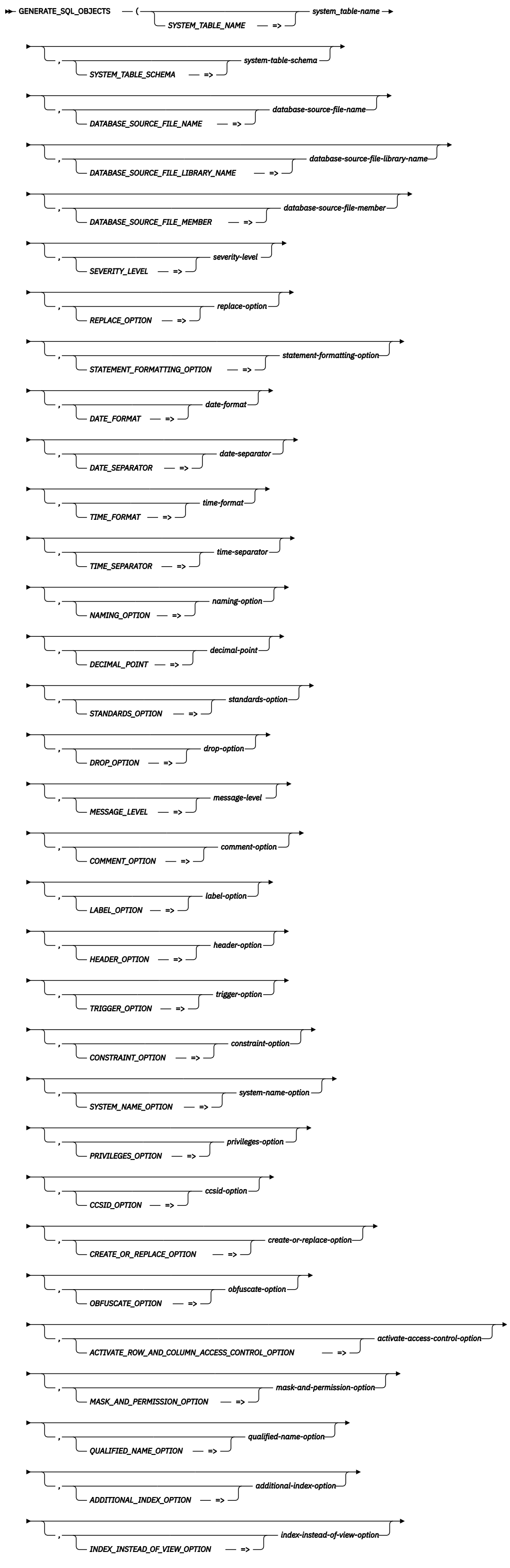
- Generate DDL for a single table and include the constraints within a CREATE OR REPLACE TABLE statement.

```
CALL QSYS2.GENERATE_SQL('EMPLOYEE', 'SAMPLE_CORPDB', 'TABLE',
                        'GENFILE', 'DDLSOURCE', 'MASTERSRC',
                        CREATE_OR_REPLACE_OPTION => '1',
                        CONSTRAINT_OPTION => '2');
```

GENERATE_SQL_OBJECTS procedure

The GENERATE_SQL_OBJECTS procedure generates the SQL data definition language (DDL) statements required to recreate a set of database objects. The results are returned in the specified database source file member or as a result set. The procedure will generate the objects in the source file member such that dependent objects are generated after depended on objects.

The database source file member will contain the generated SQL statements. If the output source file is QTEMP/Q_GENSQOBJ, the source file is returned as a result set as well.



The schema is QSYS2.

system-table-name

A character or graphic string expression that identifies the name of a table that contains the names and types of the database objects for which DDL will be generated. The system name of the table must be specified. The name is case sensitive. Delimiters must be specified if they are required. For example, a file with a name of "abc" must be specified as "abc". A file with a name of ABC must be specified in upper case. Wildcard characters are not supported.

The specified table must contain three columns that contain the names and types of the objects for which DDL will be generated. The column names of the table must be OBJECT_SCHEMA, OBJECT_NAME, and SQL_OBJECT_TYPE, in that order.

For example, create the table like this:

```
CREATE TABLE QTEMP.INORDER (OBJECT_SCHEMA VARCHAR(258),
                            OBJECT_NAME VARCHAR(258),
                            SQL_OBJECT_TYPE CHAR(10));
```

The contents of the columns must be specified according to the following rules for the corresponding parameters in the QSQGNDDL API. Each row in the table must identify an object that is distinct from every other object in the table.

OBJECT_SCHEMA
Identifies the schema name of an object for which DDL will be generated. The name must be delimited if delimiters are required in an SQL statement. This name is ignored if the specified object type is SCHEMA. *LIBL and *CURLIB are not allowed.

OBJECT_NAME
Identifies the name of an object for which DDL will be generated. The name must be delimited if delimiters are required in an SQL statement. If the object type is a FUNCTION or PROCEDURE, this name must be the specific name of the function or procedure. If TABLE or VIEW is specified for the object type, the object name must not identify an alias.

SQL_OBJECT_TYPE
Identifies the SQL object type of an object for which DDL will be generated.

ALIAS
The object is an SQL alias.

CONSTRAINT
The object attribute is a constraint.

FUNCTION
The object is an SQL function.

INDEX
The object is an SQL index.

MASK
The object is an SQL column mask.

PERMISSION
The object is an SQL row permission.

PROCEDURE
The object is an SQL procedure.

SCHEMA
The object is an SQL schema.

SEQUENCE
The object is an SQL sequence.

TABLE
The object is an SQL table or physical file.

TRIGGER
The object attribute is a trigger.

TYPE
The object is an SQL type.

VARIABLE
The object is an SQL global variable.

VIEW
The object is an SQL view or logical file.

XSR
The object is an XML schema repository object.

system-table-schema
A character or graphic string expression that identifies the name of the library of the table that contains the names and types of the database objects for which DDL will be generated. The system name of the schema must be specified. The name is case sensitive. Delimiters must be specified if they are required. For example, a schema with a name of "lib1" must be specified as "lib1". A schema with a name of LIB1 must be specified in upper case. Wildcard characters are not supported. *LIBL and *CURLIB are not allowed.

The default is QTEMP.

database-source-file-name
A character or graphic string expression that identifies the name of the source file that contains the SQL statements generated by the procedure. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. For example, a file with a name of "abc" must be specified with the surrounding quotes. A file with a name of ABC must be specified in upper case.

The record length of the specified source file must be greater than or equal to 92.

If *database-source-file-name* is not specified, `Q_GENSQOBJ` will be used.

database-source-file-library-name
A character or graphic string expression that identifies the name of the library containing the source file that contains the SQL statements generated by the procedure. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified.

If *database-source-file-library-name* is not specified, `QTEMP` will be used.

database-source-file-member
A character or graphic string expression that identifies the name of the source file member that contains the SQL statements generated by the procedure. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified.

If values are provided for *database-source-file-library-name*, *database-source-file-name*, and *database-source-file-member* the object must exist.

If *database-source-file-member* is not specified, `Q_GENSQOBJ` will be used.

severity-level
The severity level at which the operation fails. If errors occur that have a severity level greater than this value, the operation ends. The valid values are in the range 0 through 39 inclusive. Any severity 40 error will cause the procedure to fail.

If *severity-level* is not specified, 39 will be used.

replace-option
The replace option for the database source file member. The valid values are:

0
The resulting SQL statements are appended to the end of the database source file member.

1
The database source file member is cleared prior to adding the resulting SQL statements. If this option is chosen, the member may be cleared even if an error is returned from the procedure.

If *replace-option* is not specified, 1 will be used.

statement-formatting-option
The formatting option used in the generated SQL statements. The valid values are:

0
No additional formatting characters are added to the generated SQL statements.

1

Additional end-of-line characters and tab characters are added to the generated SQL statements.

If *statement-formatting-option* is not specified, 1 will be used.

date-format

The date format used for date constants in a generated SQL CREATE TABLE statement. The date format may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION.

If *date-format* is not specified, `ISO` will be used.

date-separator

The date separator used for date constants in a generated SQL CREATE TABLE statement. The date separator may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION statement.

If *date-separator* is not specified, `-` will be used.

time-format

The format used for time constants in a generated SQL CREATE TABLE statement. The time format may not apply to time constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION statement.

If *time-format* is not specified, `ISO` will be used.

time-separator

The time separator used for time constants in a generated SQL CREATE TABLE statement. The time separator may not apply to time constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, CREATE PROCEDURE, CREATE MASK, or CREATE PERMISSION statement.

If *time-separator* is not specified, `.` will be used.

naming-option

The naming convention used for qualified names in the generated SQL statements. The valid values are:

SQL

schema.table syntax

SYS

library/file syntax

If *naming-option* is not specified, SQL will be used.

decimal-point

The decimal point used for numeric constants. The valid values are:

.

Period separator

,

Comma separator

If *decimal-point* is not specified, `.` will be used.

standards-option

The standards option specifies whether the generated SQL statements should contain Db2 for i extensions or whether the statements should conform to the Db2 family SQL or to the ANS and ISO SQL standards. The valid values are:

0

Db2 for i extensions may be generated in SQL statements.

1

The generated SQL statements must conform to SQL statements common to the Db2 family.

2
: The generated SQL statements must conform to the ANSI and ISO SQL standards.

If *standards-option* is not specified, `0` will be used.

drop-option
: The drop option specifies whether DROP (or ALTER) SQL statements should be generated prior to the CREATE statement to drop the specified object. The valid values are:

0
: DROP statements should not be generated.

1
: DROP statements should be generated.

If *drop-option* is not specified, `0` will be used.

message-level
: The severity level at which the messages are generated. If errors occur that have a severity level greater than this value, a message is generated in the output. The valid values are in the range `0` through `39` inclusive. The message level must be less than or equal to the severity level.

If *message-level* is not specified, `0` will be used.

comment-option
: The comment option specifies whether COMMENT ON SQL statements should be generated if a comment exists on the specified database object. If comments are not supported by the specified database object, the comment option is ignored. The valid values are:

0
: COMMENT ON SQL statements should not be generated.

1
: COMMENT ON SQL statements should be generated. If the specified database object type is a table or view, COMMENT ON SQL statements will also be generated for columns of the table or view.

If *comment-option* is not specified, `1` will be used.

label-option
: The label option specifies whether LABEL ON SQL statements should be generated if a label exists on the specified database object. If labels are not supported by the specified database object, the label option is ignored. The valid values are:

0
: LABEL ON SQL statements should not be generated.

1
: LABEL ON SQL statements should be generated. If the specified database object type is a table or view, LABEL ON SQL statements will also be generated for columns of the table or view.

If *label-option* is not specified, `1` will be used.

header-option
: The header option specifies whether a header should be generated prior to the first generated statement. The header consists of comments that describe the version, date and time, the relational database, and some of the options used to generate the SQL statements. The valid values are:

0
: A header should not be generated.

1
: A header should be generated.

If *header-option* is not specified, `1` will be used.

trigger-option
: The trigger option specifies whether triggers should be generated when the object type is a TABLE or VIEW. The valid values are:

0
: Triggers should not be generated.

1
: Triggers should be generated.

If *trigger-option* is not specified, 1 will be used.

constraint-option
The constraint option specifies whether constraints should be generated when the object type is a TABLE. The valid values are:

0
: Constraints should not be generated.

1
: Constraints should be generated.

2
: Constraints should be generated as part of the CREATE TABLE statement.

If *constraint-option* is not specified, 1 will be used.

system-name-option
The system name option specifies whether a FOR SYSTEM NAME clause should be generated for the system name when it is different from the SQL name and the object type is an INDEX, TABLE, VIEW, or VARIABLE. The valid values are:

0
: A FOR SYSTEM NAME clause should not be generated.

1
: A FOR SYSTEM NAME clause should be generated.

If *system-name-option* is not specified, 1 will be used.

privileges-option
The privileges option specifies whether GRANT SQL statements should be generated on the specified database object. If privileges are not supported by the specified database object, the privileges option is ignored. The valid values are:

0
: GRANT SQL statements should not be generated.

1
: GRANT SQL statements should be generated.

If *privileges-option* is not specified, 1 will be used.

ccsid-option
The CCSID option specifies whether the CCSID attribute should be generated for column definitions when the object type is a TABLE. The valid values are:

0
: CCSID attribute should not be generated.

1
: CCSID attribute should be generated.

If *ccsid-option* is not specified, 1 will be used.

create-or-replace-option
The create or replace option specifies whether CREATE OR REPLACE should be generated for the specified database object on the CREATE statement. This option is ignored if the specified database object does not support CREATE OR REPLACE. The valid values are:

0
: CREATE OR REPLACE should not be generated.

1
: CREATE OR REPLACE should be generated.

If *create-or-replace-option* is not specified, `0` will be used.

obfuscate-option

The obfuscate option specifies whether an obfuscated SQL statement should be returned for SQL functions, SQL procedures, or SQL triggers that were not created using obfuscated statements. This option is ignored if the standards option is not ‘0’. This option is also ignored if the object is not an SQL function, procedure, or tirigger. This option is ignored if the object is already obfuscated. Setting Obfuscate option = 0 cannot be used as a means of obtaining the unobfuscated SQL statement for an obfuscated object. The valid values are:

0

An obfuscated statement should not be generated.

1

An obfuscated statement should be generated for SQL functions, SQL procedures, or SQL triggers.

If *obfuscate-option* is not specified, `0` will be used.

activate-access-control-option

The activate row and column access control option specifies whether an ALTER TABLE to activate row and column access control should be generated when the object type is a TABLE. This option is ignored if the standards option is not '0' or '1'. The valid values are:

0

Activate row and column access control should not be generated.

1

Activate row and column access control should be generated.

If *activate-access-control-option* is not specified, 1 will be used.

mask-and-permission-option

The mask and permission option specifies whether row permissions and column masks should be generated when the object type is a TABLE. This option is ignored if the standards option is not '0' or '1'. The valid values are:

0

Permissions and masks should not be generated.

1

Permissions and masks should be generated.

If *mask-and-permission-option* is not specified, 1 will be used.

qualified-name-option

The qualified name option specifies whether qualified or unqualified names should be generated for the specified database object. The valid values are:

0

Qualified object names should be generated. Unqualified names within the body of SQL routines will remain unqualified.

1

Unqualified object names should be generated when the a library is found which matches the database object library name. Any SQL object or column reference that is RDB qualified will be generated in its fully qualified form. For example, rdb-name.schema-name.table-name and rdb-name.schema-name.table-name.column-name references will retain their full qualification.

If *qualified-name-option* is not specified, `0` will be used.

additional-index-option

The additional index option specifies whether additional CREATE INDEX statements will be generated for DDS-created keyed physical or logical files. The valid values are:

0

Additional CREATE INDEX statements will not be generated.

1

An additional CREATE INDEX statement will be generated that matches the index for a DDS-created keyed physical file. If the physical file has a PRIMARY KEY constraint, a CREATE INDEX statement is not generated.

An additional CREATE INDEX statement will be generated that matches the index for a DDS-created keyed logical file. If a value of ‘1’ is specified for the index instead of view option, an additional CREATE INDEX statement is not generated. Additional CREATE INDEX statements will also be generated that match the join indexes of a DDS-created join logical file.

If *additional-index-option* is not specified, `0` will be used.

index-instead-of-view-option

The index instead of view option specifies whether a CREATE INDEX or CREATE VIEW statement will be generated for a DDS-created keyed logical file. The valid values are:

0

A CREATE VIEW statement will be generated.

1

A CREATE INDEX statement will be generated that matches the index for a DDS-created keyed logical file.

If *index-instead-of-view-option* is not specified, `0` will be used.

Notes

- If an error occurs while generating the DDL for an object, the source file will contain the error and processing will continue to the next object. After processing the last object, a warning SQLSTATE '01H52' will be returned.
- Objects are generated in the following order:
  1. Schemas
  2. Types
  3. Sequences
  4. Aliases
  5. Non-MQT tables and any constraints and indexes on those tables
  6. Functions
  7. Procedures
  8. Variables
  9. Views, DDS-created logical files and MQTs and any constraints and indexes on those tables
  10. Triggers
  11. Masks
  12. Permissions
  13. XSR objects

Restrictions

- One use of this procedure is to create a clone of a set of objects in another library by using `QUALIFIED_NAME_OPTION=>1`, setting the current schema and path, and then running the generated script.
  – If a depended on object is not included in the list of objects for which DDL will be generated, errors may occur when attempting to run the generated script. For example, if view V1 is based on table T1, but only V1 is specified, the attempt to run the generated script will fail because T1 was not included.
  – The QSQGNDDL API, on which this procedure is based, generates a qualified name in some cases. Thus, it may be necessary to make minor modifications to the script prior to running it. For more

information see the Qualified name option parameter in Generate Data Definition Language (QSQGNDDL) API.

- A function or procedure that has a parameter with a DEFAULT clause that references a variable, view, or MQT will not create when running the generated script. This is because variables, views, and MQTs are generated after functions and procedures. Note that references to variables, views, and MQTs within the body of function or procedure are soft dependencies and will not prevent the create.
- A variable that contains a DEFAULT clause that references a view or MQT will not create when running the generated script. This is because views and MQTs are generated after variables.

Examples

- Generate ordered DDL for the objects listed in the QTEMP.INORDER file.

```
CALL QSYS2.GENERATE_SQL_OBJECTS('INORDER', 'QTEMP');
```

RESTART_IDENTITY procedure

The RESTART_IDENTITY procedure examines the *source-table* and determines the identity column and its next value. The next value and column name are used to configure the *target-table* to use the same next value.

```
>>-- RESTART_IDENTITY -- ( -->

>-- source-schema -- , -- source-table -- , -- target-schema -- , -- target-table -- ) --><
```

The schema is QSYS2.

source-schema
A character or graphic string for the schema name containing *source-file*. It must be a system schema name.

source-table
A character or graphic string for the table name that has the identity value to copy. It must be a system table name. The table must contain an identity column.

target-schema
A character or graphic string for the schema name containing *target-table*. It must be a system schema name.

target-table
A character or graphic string for the table name that is to have its identity column value reset. It must be a system table name. The table must contain an identity column that has the same name as the identity column in *source-table*.

Example

Set the identity column in NEWTABLE to have the same next value as the identity column in OLDTABLE

```
CALL QSYS2.RESTART_IDENTITY('OLDLIB', 'OLDTABLE', 'NEWLIB', 'NEWTABLE')
```

IBM i Services

There are many system services that can be accessed through system-provided SQL views, procedures, and functions. These provide an SQL interface to access, transform, order, and subset the information without needing to code to a system API.

Application Services

These procedures and views provide interfaces that can be used in applications.

ENVIRONMENT_VARIABLE_INFO view

The ENVIRONMENT_VARIABLE_INFO view contains information about environment variables.

The values returned for the columns in the view are similar to the values returned by the WRKENVVAR CL command or Get All System-Level Environment Variables API. Refer to the API for more detailed information.

Authorization: None required.

The following table describes the columns in the view. The system name is ENV_VARS. The schema is QSYS2.

Table 67. ENVIRONMENT_VARIABLE_INFO view

Column Name	System Column Name	Data Type	Description
ENVIRONMENT_VARIABLE_TYPE	VAR_TYPE	VARCHAR(6)	The type of environment variable. **SYSTEM** Defined as a system level environment variable. **JOB** Defined as a job level environment variable. This variable and value only apply to the current connection. **PASE** Defined as an IBM® Portable Application Solutions Environment for i (PASE for i) environment variable. This variable and value only apply to the current job. PASE variables are not returned unless the PASE environment has been started.
ENVIRONMENT_VARIABLE_NAME	VAR_NAME	VARGRAPHIC(128) CCSID 1200	The name of the environment variable. If the name is longer than 128 characters, it will be truncated with no warning. If ENVIRONMENT_VARIABLE_CCSID is 65535, the content of this column is set using the job default CCSID.
ENVIRONMENT_VARIABLE_VALUE	VAR_VALUE	VARGRAPHIC(1024) CCSID 1200 Nullable	The current value of the environment variable. If the value is longer than 1024 characters, it will be truncated with no warning. If ENVIRONMENT_VARIABLE_CCSID is 65535, the content of this column is set using the job default CCSID. Contains null if there is no value.
ENVIRONMENT_VARIABLE_BINARY_NAME	VAR_BNAME	VARBINARY(128)	The name of the environment variable in binary form. This is the raw value for the name. If the name is longer than 128 characters, it will be truncated with no warning.
ENVIRONMENT_VARIABLE_BINARY_VALUE	VAR_BVALUE	VARBINARY(1024) Nullable	The current value of the environment variable. This is the raw value for the value. If the value is longer than 1024 characters, it will be truncated with no warning. Contains null if there is no value.
ENVIRONMENT_VARIABLE_CCSID	VAR_CCSID	INTEGER	The CCSID value associated with the environment variable.

Example

Look at all system level environment variables and their values for this connection:

```
SELECT ENVIRONMENT_VARIABLE_NAME, ENVIRONMENT_VARIABLE_VALUE
    FROM QSYS2.ENVIRONMENT_VARIABLE_INFO
    WHERE ENVIRONMENT_VARIABLE_TYPE = 'SYSTEM'
```

QCMDEXC procedure

The QCMDEXC procedure executes a CL command.

►► QCMDEXC ── (── *CL-command-string* ──) ─►◄

The schema is QSYS2.

CL-command-string
A character string expression containing a CL command.

The *CL-command-string* will be run as a CL command.

Examples

- Add a library to the library list.

```
CALL QSYS2.QCMDEXC('ADDLIBLE PRODLIB2');
```

- Add a library to the library list using an expression.

```
DECLARE V_LIBRARY_NAME VARCHAR(10);
SET V_LIBRARY_NAME = 'PRODLIB2';
CALL QSYS2/QCMDEXC('ADDLIBLE ' CONCAT V_LIBRARY_NAME);
```

SERVICES_INFO table

The SERVICES_INFO table returns information about system-supplied services.

The following table describes the columns in the table. The system name is SERV_INFO. The schema is QSYS2.

Table 68. SERVICES_INFO table

Column Name	System Column Name	Data Type	Description
SERVICE_CATEGORY	CATEGORY	VARCHAR(40)	Classification of the service. • APPLICATION • COMMUNICATION • DATABASE-APPLICATION • DATABASE-PERFORMANCE • DATABASE-PLAN CACHE • DATABASE-UTILITY • JAVA • JOURNAL • LIBRARIAN • MESSAGE HANDLING • PRODUCT • PTF • SECURITY • SPOOL • STORAGE • SYSTEM HEALTH • WORK MANAGEMENT
SERVICE_SCHEMA_NAME	SYS_NAME	VARCHAR(128)	Name of the schema containing the service.
SERVICE_NAME	SERVNAME	VARCHAR(128)	Name of the service.
SQL_OBJECT_TYPE	SQLTYPE	VARCHAR(15)	The type of object. • PROCEDURE • SCALAR FUNCTION • TABLE • TABLE FUNCTION • VIEW

Table 68. SERVICES_INFO table (continued)

Column Name	System Column Name	Data Type	Description
OBJECT_TYPE	OBJTYPE	VARCHAR(7) Nullable	The system object type of the service. • *FILE • *SRVPGM Contains null for procedures and functions implemented as external routines.
SYSTEM_OBJECT_NAME	SYS_ONAME	VARCHAR(10) Nullable	The system name of the service. Contains null for procedures and functions implemented as external routines.
LATEST_DB2_GROUP_LEVEL	GROUPLVL	INTEGER Nullable	The DB2 for i PTF Group level which most recently changed this service. Contains null if the service has not been enhanced in a PTF in this release.
INITIAL_DB2_GROUP_LEVEL	INITIALLVL	INTEGER Nullable	The DB2 for i PTF Group level where this service was introduced. Contains null if this service was available in the base for this release.
EARLIEST_POSSIBLE_RELEASE	MINRLS	VARCHAR(6)	The earliest release, in VxRxMx format, where a version of this service is available.
EXAMPLE	EXAMPLE	VARCHAR(5000)	An example SQL script that uses this service.

Example

Show all the available PTF services:

```
SELECT * FROM QSYS2.SERVICES_INFO
  WHERE SERVICE_CATEGORY = 'PTF'
```

DB2 PTF Group dependencies

To complement the DB2 PTF Group level information provided by the SERVICES_INFO catalog table, you can determine the DB2 PTF Group dependency level for every static SQL statement within a module, program, or service program. The QSYS2.SYSPROGRAMSTMTSTAT catalog contains one row for every static SQL statement. The DB2 PTF Group dependency information is surfaced in two columns:

SQL_DB2_GROUP_LEVEL

Indicates the use of SQL language features. For example, new SQL statements or query clauses surface as dependencies upon having a certain DB2 PTF Group level (or higher) installed before the statement can be run.

This is an SQL syntax level and is an accurate indication of the dependency level.

SERVICES_DB2_GROUP_LEVEL

Indicates the consumption of IBM i Services. For example, queries that reference DB2 for i provided views, functions, procedures, or global variables can surface possible dependencies upon having a certain DB2 PTF Group level (or higher) installed before executing the statement. If multiple services are used within a single SQL statement, the highest dependency level is returned.

The services that are instrumented are documented in "IBM i Services" on page 301 and DB2 for i Services. SQL built-in functions and built-in global variables are also tracked.

This is not an exact indication of the DB2 PTF Group that is needed. It depends on how the service is being used in your application. The information is provided based solely on the name of the service and the knowledge of when the latest enhancement was added for that service. If the name of an IBM-provided service matches an unqualified name in an SQL statement, it will be tracked as the IBM service. Based on the reported use of these services, you will need to determine whether the reported DB2 PTF Group is actually required.

To check all programs in APPLIB for potential SQL syntax and IBM i Service dependencies, execute the following query. Only programs created after the SERVICES_INFO table was introduced will report this information.

```
SELECT PROGRAM_NAME, SQL_DB2_GROUP_LEVEL, SERVICES_DB2_GROUP_LEVEL
       FROM QSYS2.SYSPROGRAMSTMTSTAT
       WHERE PROGRAM_SCHEMA = 'APPLIB' AND
             (SQL_DB2_GROUP_LEVEL IS NOT NULL OR
              SERVICES_DB2_GROUP_LEVEL IS NOT NULL);
```

To see more detailed information about which services are used in a program, including the name of each service and the DB2 PTF Group level required for the service, perform the following steps:

1. STRDBG UPDPROD(*YES)
2. Precompile your program or build your SQL procedure, function, or trigger.
   - To have informational messages written to the listing, add SET OPTION OUTPUT=*PRINT to your SQL routine or specify the OUTPUT(*PRINT) parameter on the CRTSQLxxx or RUNSQLSTM CL commands
3. For each reference to a service, message SQL7901 will be written to the joblog and, optionally, to the precompile listing.

   If you precompile with a TGTRLS of 7.1, a message will be issued for the earlier release as well with an indication of the DB2 PTF Group level that is needed on that release. If the service is not supported for a release, message SQL795B will be issued.

This information can be used to determine whether your application contains any content that might require a certain level of DB2 PTF Group. If you need to deploy your application to a different partition or an earlier release, this feedback can alert you to potential dependencies.

After you have created one or more objects using the steps above, you can query your job log to see if any messages were issued that might need to be addressed.

```
SELECT MESSAGE_ID, MESSAGE_TEXT
   FROM TABLE(QSYS2.JOBLOG_INFO('*')) X
   WHERE MESSAGE_ID IN ('SQL7901', 'SQL795B')
   ORDER BY ORDINAL_POSITION;
```

Here is one more query to help tie this information together. It will tell you the DB2 PTF Group level that is on a partition.

```
SELECT MAX(PTF_GROUP_LEVEL) AS DB2_PTF_LEVEL FROM QSYS2.GROUP_PTF_INFO
  WHERE PTF_GROUP_NAME LIKE 'SF9970%'  AND PTF_GROUP_STATUS = 'INSTALLED';
```

SET_PASE_SHELL_INFO procedure

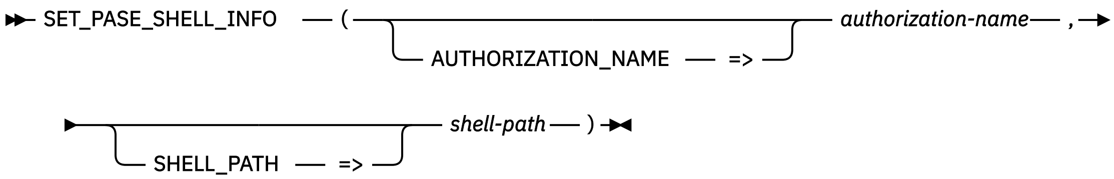
The SET_PASE_SHELL_INFO procedure provides the ability to set the path to the PASE shell for the specified user or the path to the default shell returned for users that do not have a configured shell.

The path set by this procedure is returned in the pw_shell field in struct pw from PASE APIs such as Get User Information for User Name (getpwnam) and Get User Information for User ID (getpwuid). It is also returned by the QSYS2.USER_INFO view . If a user does not have a path set, the default shell path is returned; if the default shell is not set, an empty string is returned. The pw_shell field is used by PASE applications that need to execute shells for a user, such as the OpenSSH server. The OpenSSH server will start this application as the initial program when the user logs in. If it is not set it will use /QOpenSys/usr/bin/bsh instead.

Authorization:

- If AUTHORIZATION_NAME is *CURRENT or matches the caller of this procedure, no authorization is needed.
- Otherwise the user calling this procedure must have:
  - *SECADM special authority and

– *OBJMGT and *USE to the user profile identified by AUTHORIZATION_NAME.



The schema is QSYS2.

authorization-name
A character or graphic string expression that identifies an existing user profile name. Can also be one of the following special values:

***CURRENT**
Set the current user's shell.

***DEFAULT**
Set the PASE shell to be used by any user that does not have an explicit value set. The default does not apply to IBM supplied profiles.

The default is saved in the QSYS user profile. This is equivalent to specifying 'QSYS' for *authorization-name*.

shell-path
A character or graphic string expression that specifies the path to a PASE shell. The string must begin with a forward slash (/).

If *shell-path* is blanks, the empty string, or NULL, the shell path is removed for the user. Once the path is removed, the value specified for *DEFAULT (if any) will apply to this user.

Examples

- Set the current user's shell to BASH shipped by 5733-OPS.

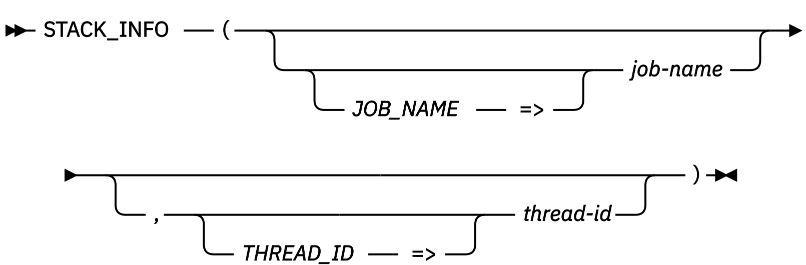
```
CALL QSYS2.SET_PASE_SHELL_INFO('*CURRENT',
                               '/QOpenSys/QIBM/ProdData/OPS/tools/bin/bash');
```

- Set the default shell to be ksh for any users that do not have an explicit shell set.

```
CALL QSYS2.SET_PASE_SHELL_INFO('*DEFAULT', '/QOpenSys/usr/bin/ksh');
```

STACK_INFO table function

The STACK_INFO table function returns one row for each entry in the call stack for either a specific thread or for every thread of the specified job. It returns information similar to what can be accessed through the Display Job (DSPJOB) CL command and the Retrieve Call Stack (QWVRCSTK) API.



The schema is QSYS2.

Authorization: The authorization ID of the statement must have *JOBCTL special authority or must be the same user profile that is running the specified *job-name*. If the authorization ID has *SERVICE special authority, the returned call stack information will include Licensed Internal Code (LIC) stack entries.

job-name
The qualified job name to return stack information for. Can contain the following special value:

*****
The current job name is used.

If *job-name* is not specified, the default is *****.

thread-id
A numeric expression indicating the thread identifier to return information for. Can contain one of the following special values:

ALL
Information for all the threads in the job is returned.

INITIAL
Information for the initial thread of the job is returned.

If *thread-id* is not specified:

- If *job-name* is *, the default is the value of the QSYS2.THREAD_ID global variable.
- Otherwise, the default is `INITIAL`.

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 69. STACK_INFO table function

Column Name	Data Type	Description
THREAD_ID	BIGINT	The identifier for the specific thread.
THREAD_TYPE	VARCHAR(6)	Specifies how the thread was initiated. **SYSTEM** The thread was initiated by the operating system. **USER** The thread was initiated by a user process. Contains the null value unless ALL was specified for the *thread-id* input parameter.
ORDINAL_POSITION	INTEGER	A unique number for each row corresponding to a thread where 1 is the first invocation entry for this thread and the highest number is the most recent invocation entry for this thread.
ENTRY_TYPE	VARCHAR(4)	The type of stack entry. **ILE** This entry returns ILE program information. The columns specific to JAVA, PASE, and LIC contain the null value. **JAVA** This entry returns JAVA information. The columns specific to ILE and OPM, PASE, and LIC contain the null value. **LIC** This entry returns Licensed Internal Code (LIC) information. The columns specific to ILE and OPM, JAVA, and PASE contain the null value. **OPM** This entry returns OPM program information. The columns specific to JAVA, PASE, and LIC contain the null value. **PASE** This entry returns PASE information. The columns specific to ILE and OPM, JAVA, and LIC contain the null value.
––– ILE and OPM information ––––––––		
PROGRAM_NAME	VARCHAR(10)	The name of the program or service program. Contains the null value if the program name is not available.
PROGRAM_LIBRARY_NAME	VARCHAR(10)	The name of the library in which the program is located. Contains the null value if the program is not located in a library or if the program library name is not available.

Table 69. STACK_INFO table function (continued)

Column Name	Data Type	Description
STATEMENT_IDENTIFIERS	VARCHAR(109)	The high-level language statement identifier. If this column contains the character representation of a number, the number is right-adjusted and padded on the left with zeros (for example, '0000000246'). If the call stack entry is for an integrated language environment (ILE) procedure, more than one statement identifier may exist. If more than one statement identifier is returned, each identifier will be ten characters long with a single blank between them. Up to ten identifiers will be returned. Returns the null value if a statement identifier cannot be determined.
REQUEST_LEVEL	INTEGER	The level of the request-processing program or procedure. Contains the null value if the program or procedure has not received a request message or incomplete information is available.
CONTROL_BOUNDARY	VARCHAR(3)	Whether a control boundary exists for a program or procedure. A control boundary is defined as any ILE call stack entry for which the immediately preceding call stack entry is for an ILE procedure or program object in a different activation group. **NO** No control boundary is active. **YES** A control boundary is active. Contains the null value if information is not available or incomplete information is available.
PROGRAM_ASP_NAME	VARCHAR(10)	The name of the auxiliary storage pool (ASP) device in which the program is located. Can contain the following special value: ***SYSBAS** The program is located in the system ASP or a basic user ASP Contains the null value if the name of the ASP cannot be determined.
PROGRAM_ASP_NUMBER	INTEGER	The numeric identifier of the ASP containing the program. **1** The program is in the system ASP. **2-32** The program is in a basic user ASP. **33-255** The program is in an independent ASP. Contains the null value if the ASP device cannot be determined.
MODULE_NAME	VARCHAR(10)	The module containing the integrated language environment (ILE) procedure. Contains the null value if this is not an ILE program or if the module name is not available.
MODULE_LIBRARY_NAME	VARCHAR(10)	The name of the library in which the module is located. Contains the null value if this is not an ILE program or if the module library name is not available.
PROCEDURE_NAME	VARCHAR(4096)	The name of the procedure at this level of the call stack. Returns the null value if this is not an ILE program or if the procedure name cannot be determined.
ACTIVATION_GROUP_NUMBER	DECIMAL(20,0)	The number of the activation group within which the program or procedure is running. This is an internal number that uniquely identifies the activation group within the job. Contains the null value if this is not an ILE program or incomplete information is available.

Table 69. STACK_INFO table function (continued)

Column Name	Data Type	Description
ACTIVATION_GROUP_NAME	VARCHAR(10)	The name of the activation group within which the program or procedure is running. Can contain the following special values: ***DFTACTGRP** The activation group does not have a specific name. The activation group is one of the default activation groups for the system. ***NEW** The activation group does not have a specific name. The activation group was created when the program was called. Contains the null value if this is not an ILE program or incomplete information is available.
MI_INSTRUCTION_NUMBER	INTEGER	The current machine instruction number in the program. Contains the null value if this is not an OPM program.
––– JAVA information ––––––––		
JAVA_LINE_NUMBER	INTEGER	The line number where the invocation was interrupted. Contains the null value if no line number can be determined.
JAVA_BYTE_CODE_OFFSET	INTEGER	The offset in bytes from the beginning of the Java method byte codes to the resume point for the invocation. Contains the null value if no Java byte code offset can be determined.
JAVA_METHOD_TYPE	VARCHAR(9)	The type of Java method. **DE** The method is a direct execution Java method. The Java method has been precompiled by the Java Transformer. **GLUE** The invocation is a Java Virtual Machine glue frame used either to perform a call from the JVM to a Java method or perform a call to a Java native method. **INTERPRET** The method is an interpreted Java method. The Java method is being interpreted by the Java Interpreter. **JIT** The method is a JIT compiled Java method. The Java method has been compiled by the Java Just In Time Compiler. **MMI** The method is a MMI interpreted Java method. The Java method is being interpreted by the Mixed Mode Java Interpreter. Contains the null value if there is no information.
JAVA_CLASS_NAME	DBCLOB(64000) CCSID 1200	The name of the Java class at this level of the call stack. Returns the null value if the class name cannot be determined.
JAVA_METHOD_NAME	DBCLOB(64000) CCSID 1200	The name of the Java method at this level of the call stack. Returns the null value if the method name cannot be determined.
JAVA_METHOD_SIGNATURE	DBCLOB(64000) CCSID 1200	The signature of the Java method at this level of the call stack. Returns the null value if the signature cannot be determined.
JAVA_FILE_NAME	DBCLOB(64000) CCSID 1200	The name of the Java file and directory that provides the location of where the Java class was loaded at this level of the call stack. If the Java class was loaded from a .jar or .zip file, then the location will be the path to and the name of the .jar or .zip file. If the class was loaded from a .class file, then the location will be the directory from which the class was loaded. Returns the null value if the file name cannot be determined.
JAVA_SOURCE_FILE_NAME	DBCLOB(64000) CCSID 1200	The name of the Java source file at this level of the call stack. Returns the null value if the source file name cannot be determined.
––– PASE information –––––––		

Table 69. STACK_INFO table function (continued)

Column Name	Data Type	Description
PASE_LINE_NUMBER	BIGINT	The line number where the invocation was interrupted. Contains the null value if no line number can be determined.
PASE_INSTRUCTION_ADDRESS	DECIMAL(20,0)	The IBM PASE for i memory address for the instruction that will run when execution resumes for the invocation.
PASE_INSTRUCTION_OFFSET	BIGINT	The offset in bytes from the beginning of the start of the procedure to the instruction that is either the suspend point for the invocation or the resume point for the invocation.
PASE_KERNEL_CODE	VARCHAR(3)	Whether the invocation is running IBM PASE for i kernel code. **NO** The current invocation is not IBM PASE for i kernel code. **YES** The current invocation is IBM PASE for i kernel code. Contains the null value if there is no information.
PASE_BIT_CODE	INTEGER	Whether the invocation is running 32-bit or 64-bit IBM PASE for i code. **32** The invocation is running 32-bit IBM PASE for i code. **64** The invocation is running 64-bit IBM PASE for i code. Contains the null value if `PASE_KERNEL_CODE` is YES.
PASE_ALTERNATE_RESUME_POINT	VARCHAR(3)	Whether the current entry is a second entry for a given invocation. This flag is only used when the system can not reliably determine which of two possible resume points will be used when an invocation resumes execution. **NO** The current invocation does not have an alternate resume point. **YES** The current invocation has an alternate resume point. Contains the null value if there is no information.
PASE_PROCEDURE_NAME	DBCLOB(4000) CCSID 1200	The name of the procedure at this level of the call stack. Returns the null value if the procedure name cannot be determined.
PASE_LOAD_MODULE_NAME	DBCLOB(1000) CCSID 1200	The name of the load module at this level of the call stack. Returns the null value if the load module name cannot be determined.
PASE_LOAD_MODULE_PATH	DBCLOB(4000) CCSID 1200	The path to the load module at this level of the call stack. Returns the null value if the load module path cannot be determined.
PASE_SOURCE_PATH_AND_FILE	DBCLOB(1000) CCSID 1200	The path and name for the source file used to create the procedure. Returns the null value if the path and name for the source file cannot be determined.
––– LIC information –––––––		
LIC_INSTRUCTION_OFFSET	BIGINT	The offset in bytes from the beginning of the start of the procedure to the instruction that is either the suspend point for the invocation or the resume point for the invocation.
LIC_PROCEDURE_NAME	VARCHAR(4096)	The name of the procedure at this level of the call stack. Returns the null value if the procedure name cannot be determined.
LIC_LOAD_MODULE_NAME	VARCHAR(64)	The name of the load module at this level of the call stack. Returns the null value if the load module name cannot be determined.

Example

- Find out whether ILE program MYPGM is on the stack for the current thread.

```
SELECT * FROM TABLE(QSYS2.STACK_INFO('*')) A
  WHERE PROGRAM_NAME = 'MYPGM';
```

- Create a table that contains the stack for all of the threads in a specific job.

```
CREATE TABLE STACK_DUMP AS (
  SELECT * FROM TABLE(QSYS2.STACK_INFO('358788/QLIWISVR/ADMIN1', 'ALL')) AS X
  ) WITH DATA;
```

Communication Services

These views and procedure provide communication information.

ENV_SYS_INFO view

The ENV_SYS_INFO view contains information about the current server.

The following table describes the columns in the view. The schema is SYSIBMADM.

Table 70. ENV_SYS_INFO view

Column Name	System Column Name	Data Type	Description
OS_NAME	OS_NAME	VARCHAR(256) Nullable	Operating system name.
OS_VERSION	OS_VERSION	VARCHAR(256) Nullable	Operating system version.
OS_RELEASE	OS_RELEASE	VARCHAR(256) Nullable	Operating system release.
HOST_NAME	HOST_NAME	VARCHAR(256) Nullable	Name of the system.
TOTAL_CPUS	TOTAL_CPUS	INTEGER Nullable	The maximum number of virtual processors defined within the LPAR configuration.
CONFIGURED_CPUS	CONFIGCPUS	INTEGER Nullable	The number of virtual processors currently available to the partition.
CONFIGURED_MEMORY	CONFIGMEM	BIGINT Nullable	Total amount of configured memory on the system, in megabytes.
TOTAL_MEMORY	TOTAL_MEM	INTEGER Nullable	Total amount of memory on the system, in megabytes.

Example

Return information about the current server.

```
SELECT * FROM SYSIBMADM.ENV_SYS_INFO
```

NETSTAT_INFO view

The NETSTAT_INFO view returns information about IPv4 and IPv6 network connections.

The values returned for the columns in the view are closely related to the values returned by List Network Connections API and Retrieve Network Connection Data API. Refer to the APIs for more detailed information.

The following table describes the columns in the view. The system name is NS_INFO. The schema is QSYS2.

Table 71. NETSTAT_INFO view

Column Name	System Column Name	Data Type	Description
CONNECTION_TYPE	CONN_TYPE	CHAR(4)	The type of connection. **IPV4** The connection is an IPv4 connection. **IPV6** The connection is an IPv6 connection.
REMOTE_ADDRESS	RMT_ADDR	VARCHAR(45)	The internet address of the remote host. For IPv4: • The address is in IPv4 address format. A value of 0.0.0.0 indicates that either the system is waiting for a connection to open or that a UDP socket is being used. A value of 0 means that the connection is a listening or UDP socket so this field does not apply. For IPv6: • The address is in IPv6 address format. A value of :: means that the connection is a listening socket so this field does not apply.
REMOTE_PORT	RMT_PORT	INTEGER	The remote host port number. A value of 0 means that the connection is a listening or UDP socket, so this field does not apply.
REMOTE_PORT_NAME	RMT_NAME	VARGRAPHIC(14) CCSID 1200 Nullable	The remote host well-known port name or the name from the service table entry. Contains null if there is no well-known port name.
LOCAL_ADDRESS	LOCAL_ADDR	VARCHAR(45)	The local address of this connection on this system. For IPv4: • The address is in IPv4 address format. A value of 0.0.0.0 indicates that either the system is waiting for a connection to open or that a UDP socket is being used. For IPv6: • The address is in IPv6 address format. A value of :: means the local application specified that any local internet address can be used.
LOCAL_PORT	LOCAL_PORT	INTEGER	The local system port number.
LOCAL_PORT_NAME	LOCAL_NAME	VARGRAPHIC(14) CCSID 1200 Nullable	The local system well-known port name or the name from the service table entry. Contains null if there is no well-known port name.
PROTOCOL	PROTOCOL	VARCHAR(3)	Identifies the type of connection protocol. **TCP** A Transmission Control Protocol (TCP) connection or socket. **UDP** A User Datagram Protocol (UDP) socket.

Table 71. NETSTAT_INFO view (continued)

Column Name	System Column Name	Data Type	Description
TCP_STATE	STATE	VARCHAR(12) Nullable	The state of the connection. **CLOSED** This connection has ended. **CLOSE-WAIT** Waiting for an end connection request from the local user. **CLOSING** Waiting for an end connection request acknowledgment from the remote host. **ESTABLISHED** The normal state in which data is transferred. **FIN-WAIT-1** Waiting for the remote host to acknowledge the local system request to end the connection. **FIN-WAIT-2** Waiting for the remote host request to end the connection. **LAST-ACK** Waiting for the remote host to acknowledge an end connection request. **LISTEN** Waiting for a connection request from any remote host. **SYN-RECEIVED** Waiting for a confirming connection request acknowledgment. **SYN-SENT** Waiting for a matching connection request after having sent a connection request. **TIME-WAIT** Waiting to allow the remote host enough time to receive the local system's acknowledgment to end the connection. Contains null if PROTOCOL is UDP.
IDLE_TIME	IDLE_TIME	DECIMAL(19,3)	The length of time, in seconds, since the last activity on this connection.
BIND_USER	BIND_USER	VARCHAR(10)	The user profile of the job on the local system which first performed a sockets API bind() of the socket.
BYTES_SENT_REMOTELY	BYTES_OUT	BIGINT	The number of bytes sent to the remote host.
BYTES_RECEIVED_LOCALLY	BYTES_IN	BIGINT	The number of bytes received from the remote host.
NETWORK_CONNECTION_TYPE	NET_TYPE	VARCHAR(4)	The type of connection or socket. ***TCP** Identifies a transmission control protocol (TCP) connection socket. ***UDP** Identifies a User Datagram Protocol (UDP) socket. For IPv4, the following additional value can be returned. ***IPS** Identifies an Internet Protocol (IP) over SNA connection or socket.
CONNECTION_OPEN_TYPE	OPN_TYPE	VARCHAR(7) Nullable	The type of open for the connection. **ACTIVE** The local system opens the connection. **PASSIVE** A remote host opens the connection. Contains null if PROTOCOL is UDP.
NUMBER_OF_ASSOCIATED_JOBS	NUM_JOBS	INTEGER	The number of jobs associated with this connection.
LINE_DESCRIPTION	LINE_DES	VARCHAR(10) Nullable	The local system line description associated with this connection. Contains null if this is an IPv4 connection or if the connection is not bound to a link local unicast interface.

Table 71. NETSTAT_INFO view (continued)

Column Name	System Column Name	Data Type	Description
VIRTUAL_LAN_ID	LAN_ID	VARCHAR(4) Nullable	The virtual LAN identifier associated with this connection. Can also contain the following special value: **NONE** No virtual LAN identifier is associated with this connection. Contains null if this is an IPv4 connection or if the connection is not bound to a link local unicast interface.
CONNECTION_TRANSPORT_LAYER	CNNTRANSPT	VARCHAR(5)	The transport that a connection is using. Values are: • **IPS** • **TCPIP**
IP_OPTIONS	IP_OPTIONS	BINARY(40) Nullable	The hex value of IP datagram options that may have been specified for a connection. Contains null if this is an IPv6 connection or if no IP datagram options have been specified.
ROUND_TRIP_TIME	ROUND_TRIP	BIGINT Nullable	The smoothed round-trip time interval in milliseconds. This is a measure of the time required for a segment on the connection to arrive at its destination, to be processed, and to return an acknowledgment to the client. Contains null if PROTOCOL is UDP.
ROUND_TRIP_VARIANCE	ROUND_VAR	BIGINT Nullable	The variance in milliseconds from the previous round-trip time. Contains null if PROTOCOL is UDP.
CURRENT_RETRANSMISSIONS	CT_RETRANS	BIGINT Nullable	The number of times the local system retransmitted the current segment without receiving an acknowledgment. Contains null if PROTOCOL is UDP.
TOTAL_RETRANSMISSIONS	TL_RETRANS	BIGINT Nullable	The total number of times the local system retransmitted a segment because an acknowledgement was not received. This is a cumulative count of all segments resent during the entire time the connection has been active. Contains null if PROTOCOL is UDP.
TCP_CONNECTIONS_CURRENTLY_ESTABLISHED	TCPCONN	BIGINT Nullable	The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT. Contains null if PROTOCOL is UDP.
TCP_ACTIVE_OPENS	TCPACTOPN	BIGINT Nullable	The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state. This number is an indication of the number of times this local system opened a connection to a remote system. Contains null if PROTOCOL is UDP.
TCP_PASSIVE_OPENS	TCPPSVOPN	BIGINT Nullable	The number of times TCP connections have made a direct transition to the SYN-RECEIVED state from the LISTEN state. This number is an indication of the number of times a remote system opened a connection to this system. Contains null if PROTOCOL is UDP.
TCP_FAILED_OPENS	TCPFAILOPN	BIGINT Nullable	The total number of times TCP connections have made direct transitions to a CLOSED state from either the SYN-SENT state or the SYN-RECEIVED state and/or to LISTEN from SYN-RECEIVED. Contains null if PROTOCOL is UDP.
TCP_ESTABLISHED_AND_THEN_RESET	TCPESTRST	BIGINT Nullable	The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state. Contains null if PROTOCOL is UDP.
TCP_SEGMENTS_SENT	TCPSEGSENT	BIGINT Nullable	The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets. Contains null if PROTOCOL is UDP.
TCP_SEGMENTS_RETRANSMITTED	TCPSEGRTRN	BIGINT Nullable	The number of TCP segments transmitted containing one or more previously transmitted octets. Contains null if PROTOCOL is UDP.

Table 71. NETSTAT_INFO view (continued)

Column Name	System Column Name	Data Type	Description
TCP_SEGMENTS_RESET	TCPSEGRST	BIGINT Nullable	The number of TCP segments sent containing the RST flag. Contains null if PROTOCOL is UDP.
TCP_SEGMENTS_RECEIVED	TCPSEGRCV	BIGINT Nullable	The total number of segments received, including those received in error. This count includes segments received on currently established connections. Contains null if PROTOCOL is UDP.
TCP_SEGMENTS_RECEIVED_ ERROR	TCPSEGRCVE	BIGINT Nullable	The total number of segments received in error (for example, bad TCP checksums). Contains null if PROTOCOL is UDP.
OUTGOING_BYTES_BUFFERED	BYTES_OUTB	BIGINT Nullable	The current number of bytes that an application has requested to send, but TCP has not yet sent. If TCP has sent the bytes to the remote system but has not yet received an acknowledgment, the bytes are considered 'not sent'. They are included in this count. Contains null if PROTOCOL is UDP.
USER_SEND_NEXT	USRSNDNXT	BIGINT Nullable	The sequence number of the next byte of data to be sent by the client application. Contains null if PROTOCOL is UDP.
SEND_NEXT	SEND_NEXT	BIGINT Nullable	The sequence number of the next byte of data that the local TCP application sends to the remote TCP application. Contains null if PROTOCOL is UDP.
SEND_UNACKNOWLEDGED	SNDUNACK	BIGINT Nullable	The sequence number of the last segment sent that was not acknowledged. This is the smallest sequence number of the send window. Contains null if PROTOCOL is UDP.
OUTGOING_PUSH_NUMBER	OUTPSHNBR	BIGINT Nullable	The sequence number of the last byte of push data in the outgoing stream. This value is zero if no push data is in the outgoing data stream. Contains null if PROTOCOL is UDP.
OUTGOING_URGENCY_NUMBER	OUTURGNBR	BIGINT Nullable	The sequence number of the last byte of urgent data in the outgoing data stream. This value is zero if no urgent data is in the outgoing data stream. Contains null if PROTOCOL is UDP.
OUTGOING_WINDOW_NUMBER	OUTWINNBR	BIGINT Nullable	The largest sequence number in the send window of the connection. The local TCP application cannot send data bytes with sequence numbers greater than the outgoing window number. Contains null if PROTOCOL is UDP.
INCOMING_BYTES_BUFFERED	BYTES_INB	BIGINT Nullable	The current number of bytes that are received and buffered by TCP. These bytes are available to be read by an application. Contains null if PROTOCOL is UDP.
RECEIVE_NEXT	RCVNEXT	BIGINT Nullable	The next sequence number the local TCP is expecting to receive. Contains null if PROTOCOL is UDP.
USER_RECEIVE_NEXT	USRRCVNXT	BIGINT Nullable	The sequence number of the next byte to be passed to the application by TCP. Contains null if PROTOCOL is UDP.
INCOMING_PUSH_NUMBER	INPSHNBR	BIGINT Nullable	The sequence number of the last byte of pushed data in the incoming data stream. This value is zero if no push data is in the incoming data stream. Contains null if PROTOCOL is UDP.
INCOMING_URGENCY_NUMBER	INURGNBR	BIGINT Nullable	The sequence number of the last byte of urgent data in the incoming data stream. This value is zero if no urgent data is in the incoming data stream. Contains null if PROTOCOL is UDP.

Table 71. NETSTAT_INFO view (continued)

Column Name	System Column Name	Data Type	Description
INCOMING_WINDOW_NUMBER	INWINNBR	BIGINT Nullable	The largest sequence number in the incoming window of this connection. Data bytes in the incoming stream having sequence numbers larger than this number are not accepted. Contains null if PROTOCOL is UDP.
MAXIMUM_WINDOW_SIZE	MAXWINSIZ	BIGINT Nullable	The largest size of the send window, in bytes, during the entire time the connection has been active. Contains null if PROTOCOL is UDP.
CURRENT_WINDOW_SIZE	CURWINSIZ	BIGINT Nullable	The current send window size in bytes. Contains null if PROTOCOL is UDP.
LAST_UPDATE	LSTUPD	BIGINT Nullable	The sequence number of the incoming segment used for the last window update that occurred on the connection. Contains null if PROTOCOL is UDP.
LAST_UPDATE_ACKNOWLEDGED	LSTUPDACK	BIGINT Nullable	The acknowledgment number of the incoming segment used for the last window update that occurred on the connection. Contains null if PROTOCOL is UDP.
CONGESTION_WINDOW	CONGESTWIN	BIGINT Nullable	The number of segments that are sent on the next transmission. If an acknowledgment is received, the number is increased. If an acknowledgment is not received, the number is reset to the smallest allowable number. Contains null if PROTOCOL is UDP.
SLOW_START_THRESHOLD	SLWSTRTHR	BIGINT Nullable	The value of the slow-start threshold. Contains null if PROTOCOL is UDP.
MAXIMUM_SEGMENT_SIZE	MAXSEGSIZ	BIGINT Nullable	The size in bytes of the largest segment that may be transmitted on this connection. Contains null if PROTOCOL is UDP.
INITIAL_SEND_SEQUENCE_ NUMBER	SNDSEQNBR	BIGINT Nullable	The first sequence number sent on this connection. Contains null if PROTOCOL is UDP.
INITIAL_RECEIVE_SEQUENCE_ NUMBER	RCVSEQNBR	BIGINT Nullable	The first sequence number received on this connection. Contains null if PROTOCOL is UDP.
UDP_DATAGRAMS_SENT	UDPSENT	BIGINT Nullable	The total number of UDP datagrams sent from all connections since TCP/IP was started. Contains null if PROTOCOL is TCP.
UDP_DATAGRAMS_RECEIVED	UDPRCV	BIGINT Nullable	The total number of UDP datagrams received, including those received in error. This count includes datagrams received on currently established connections. Contains null if PROTOCOL is TCP.
UDP_DATAGRAMS_NOT_ DELIVERED_PORT_ NOT_FOUND	UDPNDPNF	BIGINT Nullable	The total number of received UDP datagrams for UDP users for which there was no application at the destination port. Contains null if PROTOCOL is TCP.
UDP_DATAGRAMS_NOT_ DELIVERED_OTHER	UDPNDOTHER	BIGINT Nullable	The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port. Contains null if PROTOCOL is TCP.
SOCKET_STATE	SOCSTATE	VARCHAR(13)	The current state of the socket. Values are: • **BOUND** • **CONNECTED** • **CONNECTING** • **DISCONNECTED** • **ERROR** • **LISTENING** • **UNBOUND** • **UNINITIALIZED**

Table 71. NETSTAT_INFO view (continued)

Column Name	System Column Name	Data Type	Description
SOCKET_BROADCAST	SOCBROAD	VARCHAR(3) Nullable	Indicates if messages can be sent to the broadcast address. **NO** Messages cannot be sent to the broadcast address. **YES** Messages can be sent to the broadcast address. Contains null if value is not specified or if socket is not an address family of AF_INET and type SOCK_DGRAM or SOCK_RAW.
SOCKET_BYPASS_ROUTE	SOCBYPASS	VARCHAR(3) Nullable	Indicates if the normal routing mechanism is being bypassed. **NO** The normal routing mechanism is being used. **YES** The normal routing mechanism is being bypassed. Contains null if value is not specified or if socket is not an address family of AF_INET or AF_INET6.
SOCKET_DEBUG	SOCDEBUG	VARCHAR(3) Nullable	Indicates if low-level debugging is active. **NO** Low-level debugging is not active. **YES** Low-level debugging is active. Contains null if value is not specified.
SOCKET_ERROR	SOCERROR	INTEGER Nullable	Indicates if there any pending errors in the socket. A value of zero indicates no pending errors. Otherwise, the value indicates the error number. Contains null if value is not specified.
SOCKET_KEEP_ALIVE	SOCALIVE	VARCHAR(3) Nullable	Indicates if the connection is being kept up by periodic transmissions. **NO** The connection is not being kept up by periodic transmissions. **YES** The connection is being kept up by periodic transmissions. Contains null if value is not specified or if socket is not an address family of AF_INET or AF_INET6 and type SOCK_STREAM.
SOCKET_LINGER	SOCLINGER	VARCHAR(3) Nullable	Indicates whether the system attempts to deliver any buffered data or if the system discards it when a close() is issued. **NO** The system attempts to send buffered data with an infinite wait time. **YES** The system attempts to send buffered data for SOCKET_LINGER_TIME seconds. If the data is not deliverable within that period of time, it is discarded. Contains null if value is not specified.
SOCKET_LINGER_TIME	SOCLTIME	BIGINT Nullable	The time, in seconds, the system will wait to send buffered data. Contains null if value is not specified or if SOCKET_LINGER is NO.
SOCKET_OUT_OF_BAND_DATA	SOCOUTBAND	VARCHAR(3) Nullable	Indicates if out-of-band data is received inline with normal data. **NO** Out-of-band data is not received inline with normal data. **YES** Out-of-band data is received inline with normal data. Contains null if value is not specified or if socket is not an address family of AF_INET or AF_INET6.
SOCKET_RECEIVE_BUFFER_SIZE	SOCRCVBUF	BIGINT Nullable	The size of the receive buffer. Contains null if value is not specified.

Table 71. NETSTAT_INFO view (continued)

Column Name	System Column Name	Data Type	Description
SOCKET_RECEIVE_LOW_WATER_ MARK_SIZE	SOCRCVSZ	BIGINT Nullable	The size of the receive low-water mark. The default size is 1. Contains null if value is not specified or if socket is not type SOCK_STREAM.
SOCKET_REUSE_ADDRESS	SOCREUSE	VARCHAR(3) Nullable	Indicates if the local socket address can be reused. **NO** The local socket address cannot be reused. **YES** The local socket address can be reused. Contains null if value is not specified or if socket is not an address family of AF_INET or AF_INET6 and type SOCK_STREAM or SOCK_DGRAM.
SOCKET_SEND_BUFFER_SIZE	SOCSENDBUF	BIGINT Nullable	The size of the send buffer. Contains null if value is not specified.
SOCKET_TYPE	SOCTYPE	VARCHAR(14) Nullable	The socket type. Values are: **SOCK_DGRAM** Datagram type. **SOCK_RAW** Raw type. **SOCK_SEQPACKET** Sequential packet type. **SOCK_STREAM** Stream type. Contains null if value is not specified.
SOCKET_LOOPBACK	SOCLOOPBK	VARCHAR(3) Nullable	Indicates if the loopback feature is being used. **NO** The loopback feature is not being used. **YES** The loopback feature is being used. Contains null if value is not specified.
SOCKET_RECEIVE_TIMEOUT	SOCRCVTO	BIGINT Nullable	The receive timeout value. Contains null if value is not specified.
SOCKET_SEND_LOW_WATER_ MARK_SIZE	SOCSENDSZ	BIGINT Nullable	The size of the send low-water mark. Contains null if value is not specified.
SOCKET_SEND_TIMEOUT	SOCSENDTO	BIGINT Nullable	The send timeout value. Contains null if value is not specified.

Example

Return information about all network connections for user QLWISVR.

```
SELECT * FROM QSYS2.NETSTAT_INFO
  WHERE BIND_USER = 'QLWISVR'
```

Related information

Internet Protocol version 6

NETSTAT_INTERFACE_INFO view

The NETSTAT_INTERFACE_INFO view returns information about IPv4 and IPv6 interfaces.

The values returned for the columns in the view are closely related to the values returned by List Network Interfaces API. Refer to the API for more detailed information.

The following table describes the columns in the view. The system name is NS_INTER. The schema is QSYS2.

Table 72. NETSTAT_INTERFACE_INFO view

Column Name	System Column Name	Data Type	Description
CONNECTION_TYPE	CONN_TYPE	CHAR(4)	The type of connection. **IPV4** The connection is an IPv4 connection. **IPV6** The connection is an IPv6 connection.
INTERNET_ADDRESS	INT_ADDR	VARCHAR(45)	The internet address of the interface. For IPv4: • The address is in IPv4 address format. Can contain the special value: ***IP4DHCP** The interface has been configured to use DHCP to obtain an IPv4 address. For IPv6: • The address is in IPv6 address format. Can contain the special value: ***IP6SAC** This interface will use Stateless Address Auto-configuration to obtain an IPv6 address.
NETWORK_ADDRESS	NET_ADDR	VARCHAR(45)	The internet address of the IP network or subnetwork to which the interface is attached. For IPv4: • The address is in IPv4 address format. For IPv6: • The address is in IPv6 address format.
SUBNET_MASK	SUBNET_MSK	VARCHAR(15) Nullable	The subnet mask for the network, subnet, and host address fields of the internet address that defines the subnetwork for an interface. Contains null if this is an IPv6 connection.
PREFIX_LENGTH	PRE_LEN	INTEGER Nullable	The prefix length defines how many bits of the IPv6 internet address are in the prefix. It specifies how many of the left-most bits of the address make up the prefix. The prefix length is used to generate network and host addresses. Contains null if this is an IPv4 connection.
LINE_DESCRIPTION	LINE_DES	VARCHAR(10)	The name of the communications line description that identifies the physical network associated with an interface. Can contain the following special values: ***LOOPBACK** This is the loopback interface. Processing associated with a loopback interface does not extend to a physical line. ***VIRTUALIP** The virtual interface is a circuitless interface. For IPv4, the following additional values can be returned. ***IPS** The interface is used by Internet Protocol (IP) over SNA. ***OPC** The interface is attached to the optical bus (OptiConnect).

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
INTERFACE_LINE_TYPE	LINE_TYPE	VARCHAR(6)	The type of line used by the interface. **ASYNC** Asynchronous communications protocol. **DDI** Distributed Data Interface protocol. **ELAN** Ethernet local area network protocol. **FR** Frame relay network protocol. **L2TP** Layer Two Tunneling protocol. **PPP** Point-to-Point protocol. **PPPOE** Point-to-Point over Ethernet protocol. **TDLC** Twinaxial Datalink Control. Used for TCP/IP over Twinax. **TRLAN** Token-ring local area network protocol. **VETH** Virtual Ethernet protocol. **WLS** Wireless local area network protocol. **X25** X.25 protocol. Can also contain one of the following special values: **ERROR** A system error other than those for NOTFND was received while trying to determine the link type for an interface. **NONE** Line is not defined. This value is used for the following interfaces: *LOOPBACK, *VIRTUALIP, *OPC. There is no line type value for this interface. **NOTFND** Not found. The line description object for this interface cannot be found. **OTHER** An Internet Protocol (IP) over SNA interface.

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
INTERFACE_STATUS	STATUS	VARCHAR(12)	The current status of the logical interface. **ACTIVE** The interface has been started and is running. **ENDING** The operating system is processing the request to end this interface. **FAILED** The line description associated with this interface has entered the failed state. **FAILED (TCP)** An error was detected in the IBM TCP/IP Licensed Internal Code. **INACTIVE** The interface has not been started. **RCYCNL** A hardware failure has occurred and the line description associated with this interface is in the recovery canceled (RCYCNL) state. **RCYPND** An error with the physical line associated with this interface was detected by the system. The line description associated with this interface is in the recovery pending (RCYPND) state. **STARTING** The operating system is processing the request to start this interface. For IPv4, the following additional values can be returned. **ACQUIRING** The operating system is attempting to obtain an IP address from a Dynamic Host Configuration Protocol (DHCP) server. **DOD** This interface is being used for Point-to-Point (PPP) Dial-on-Demand.
INTERFACE_SOURCE	SOURCE	VARCHAR(9) Nullable	Specifies how this interface was added to the protocol stack. **LOOPBACK** The interface was added by the protocol stack as the loopback address. **STATELESS** The interface was added by IPv6 address autoconfiguration. **STATEFUL** The interface was added by Dynamic Host Configuration Protocol version 6 (DHCPv6) configuration. **MANUAL** The interface was added by manual configuration. Contains null if this is an IPv4 connection.
SERVICE_TYPE	SRVC_TYPE	VARCHAR(9) Nullable	The type of service that defines how the internet hosts and routers should make trade-offs between throughput, delay, reliability, and cost. **MAXRLB** A higher level of effort to ensure delivery is important for datagrams with the maximize reliability indication. **MAXTHRPUT** High data rate is important for datagrams with the maximize throughput indication. **MINCOST** Lower cost is important for datagrams with the minimize monetary cost indication. **MINDELAY** Prompt delivery is important for datagrams with the minimize delay indication. **NORMAL** Normal service is used for delivery of datagrams. **OTHER** An Internet Protocol (IP) over SNA interface. Contains null if this is an IPv6 connection.

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
VIRTUAL_LAN_ID	LAN_ID	VARCHAR(4)	The virtual LAN to which this interface belongs according to IEEE standard 802.1Q. Can also contain the following special value: **NONE** The interface does not belong to a virtual LAN.
MAXIMUM_TRANSMISSION_ UNIT	MTU	VARCHAR(10)	The maximum transmission unit (MTU) value specified for this interface. Either an integer value or this special value: **LIND** The interface is not currently active and the MTU was specified as *LIND. For IPv4, the following additional value can be returned. **OTHER** An Internet Protocol (IP) over SNA interface.
CONFIGURED_MAXIMUM_ TRANSMISSION_UNIT	CFG_MTU	VARCHAR(10)	The configured maximum transmission unit value specified for this interface. Either an integer value or this special value: **LIND** The interface is not currently active and the MTU was specified as *LIND.
AUTOSTART	AUTOSTART	VARCHAR(3)	Specifies whether the interface is automatically started when the protocol stack is activated. **NO** This interface is not automatically started. **YES** This interface is automatically started.
DAD_MAX_TRANSMITS	DAD_MAX	BIGINT Nullable	The maximum number of duplicate address detection (DAD) transmissions the protocol stack will send out on this interface. Contains null if this is an IPv4 connection.
HOST_ADDRESS	HOST_ADDR	VARCHAR(45)	Host portion of the internet address. For IPv4: • Host portion of the Internet address, in dotted decimal notation, as determined by the subnet mask specified for this interface. For IPv6: • Host portion of the Internet address, in IPv6 address format, as determined by the prefix length configured for this interface.
DIRECTED_BROADCAST_ ADDRESS	DIRBRDADR	VARCHAR(15) Nullable	The Internet address, in dotted decimal notation, used to broadcast to all systems attached to the same network or subnetwork as this interface. Contains null if this is an IPv6 connection or if interface is attached to a network that does not support a broadcast operation.
ASSOCIATED_LOCAL_ INTERFACE	ASCLCLINT	VARCHAR(15) Nullable	The Internet address, in dotted decimal notation, of the local interface that has been associated with this interface. Contains null if this is an IPv6 connection or if no association has been made between this interface and another local interface.
CHANGE_STATUS	CHGSTS	VARCHAR(6) Nullable	The status of the most recent change to this interface in the dynamic tables used by the TCP/IP protocol stack. **ADD** Add interface request processed. **CHANGE** Change interface request processed. **END** End interface request processed. **START** Start interface request processed. Contains null if this is an IPv6 connection.

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
PACKET_RULES	PKT_RULES	VARCHAR(17) Nullable	The kind of packet rules data available for this line. **NONE** No filters and no NAT are loaded for this line. **NAT** NAT is enabled for this line. **FILTERS** Filters are defined for this line. **NAT_FILTERS** NAT enabled and filters defined for this line. **FILTERS_IPSEC** Filters and IPSec filters are defined for this line. **NAT_FILTERS_IPSEC** NAT enabled and Filters and IPsec filters defined for this line. Contains null if packet rules data is unknown.
INTERFACE_TYPE	TYPE	VARCHAR(12) Nullable	The interface type: **BROADCAST** Broadcast capable. **NONBROADCAST** Non-broadcast capable. **UNNUMBERED** Unnumbered network. Contains null if this is an IPv6 connection.
NETWORK_FULL_NAME	NET_FNAME	VARCHAR(24) Nullable	The complete name of the network that this interface is a part of. Contains null if this is an IPv6 connection or if there is no network name.
INTERFACE_FULL_NAME	FNAME	VARCHAR(24) Nullable	The complete interface name. Contains null if this is an IPv6 connection or if there is no interface name.
ALIAS_NAME	ALIAS_NAME	VARGRAPHIC(50) CCSID 1200 Nullable	Name given to the interface to use as an alternate to the IP address. Contains null if there is no alias name.
INTERFACE_TEXT	LABEL	VARGRAPHIC(50) CCSID 1200 Nullable	Description of the interface. Contains null if there is no interface description.
DHCP_CREATED	DHCPCRT	VARCHAR(3) Nullable	Specifies whether this interface was created using Dynamic Host Configuration Protocol (DHCP). **NO** This interface was not created using DHCP. **YES** This interface was created using DHCP. Contains null if this is an IPv6 connection.
DHCP_DYNAMIC_DNS_ UPDATES	DHCPDYNDNS	VARCHAR(3) Nullable	Specifies whether dynamic updates to Domain Name System (DNS) tables are enabled or not. **NO** DNS updates are disabled. **YES** DNS updates are enabled. Contains null if this is an IPv6 connection or if interface was not created by DHCP.
DHCP_LEASE_EXPIRATION	DHCPLEXP	TIMESTAMP(0) Nullable	The timestamp when the DHCP lease will expire. Contains null if this is an IPv6 connection or if interface was not created by DHCP.
DHCP_LEASE_OBTAINED	DHCPLOBT	TIMESTAMP(0) Nullable	The timestamp when the DHCP lease was obtained or renewed. Contains null if interface was not created by DHCP.

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
DHCP_USE_UNIQUE_ID	DHCPUSEUID	VARCHAR(3) Nullable	Whether the DHCP unique identifier (DUID) is used as the client identification for the Dynamic Host Configuration Protocol (DHCP). **NO** The hardware (MAC) address is used for the client ID. **YES** The DHCP unique identifier is used for the client ID or this is an IPv6 connection. Contains null if interface was not created by DHCP.
DHCP_SERVER_UNIQUE_ID	DHCPSRVUID	VARCHAR(30) Nullable	Specifies the DHCP unique identifier (DUID) of the DHCP server from which the IP address was obtained. Contains null if this is an IPv4 connection or if interface was not created by DHCP.
DHCP_SERVER_ADDRESS	DHCPSRVADD	VARCHAR(15) Nullable	The Internet address, in dotted decimal notation, of the DHCP server from which the DHCP lease was obtained or renewed. Contains null if this is an IPv6 connection or if interface was not created by DHCP.
PREFERRED_INTERFACE_ DEFAULT_ROUTE	PREFDFTRTE	VARCHAR(3) Nullable	This field describes whether the preferred proxy interfaces are based on the system's default route. **NO** The default route is not used to determine the preferred interface. **YES** The default route is used to determine the preferred interface. Contains null if this is an IPv6 connection.
PREFERRED_INTERFACE_LIST	PREFIFCLST	VARCHAR(159) Nullable	A list of up to 10 preferred interface internet addresses. Each internet address within the preferred interface list is given in dotted decimal notation. When there is more than one internet address, a single blank separates the addresses. Contains null if this is an IPv6 connection or if a preferred interface list is not being used.
PREFERRED_PHYSICAL_LINE_ LIST	PREFLINLST	VARCHAR(159) Nullable	A list of up to 10 preferred physical line list entries. Each entry in the list is formatted as `LINE_DESCRIPTION:VIRTUAL_LAN_ID`. The line description can be up to 10 characters long. The virtual LAN ID can be up to 4 characters long. When there is more than one preferred physical line, a single blank separates entries. Contains null if this is an IPv4 connection.
ADDRESS_TYPE	ADDR_TYPE	VARCHAR(9) Nullable	The type of IPv6 address that is assigned to this network interface. **ANYCAST** An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to an anycast address is delivered to one of the interfaces identified by that address (the "nearest" one, according to the routing protocols' measure of distance). **MULTICAST** An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address. **UNICAST** An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address. Contains null if this is an IPv4 connection.
ADDRESS_CLASS	ADDR_CLASS	VARCHAR(9) Nullable	The class of IPv6 address that is assigned to this network interface. **PUBLIC** The interface is a public one. **TEMPORARY** The interface is a temporary one used for privacy extensions. Contains null if this is an IPv4 connection.

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
ADDRESS_PREFERRED_LIFETIME	ADDPLIFE	BIGINT Nullable	The length of time that a "valid" address is preferred, in seconds. A negative value indicates that the address preferred lifetime expired that number of seconds ago. Contains null if this is an IPv4 connection or if ADDRESS_CLASS is not TEMPORARY.
ADDRESS_VALID_LIFETIME	ADDVLIFE	BIGINT Nullable	The length of time, in seconds, that an address remains in a "valid" state. A negative value indicates that the address valid lifetime expired that number of seconds ago. Contains null if this is an IPv4 connection or if ADDRESS_CLASS is not TEMPORARY.
ADDRESS_PREFERRED_ LIFETIME_EXPIRATION	ADDPFRLE	TIMESTAMP(0) Nullable	The timestamp when this address will no longer be in the preferred state. If the timestamp is in the future, the address is still preferred. If the timestamp is in the past, then this address is no longer preferred. Contains null if this is an IPv4 connection or if ADDRESS_CLASS is not TEMPORARY.
ADDRESS_VALID_LIFETIME_ EXPIRATION	ADDVLDLE	TIMESTAMP(0) Nullable	The timestamp when this address will expire or did expire. If the timestamp is in the future, the address has not expired yet. If the timestamp is in the past, then this address has expired and is still being returned for a short period of time to indicate that the interface ceased to function because its valid lifetime expired. Contains null if this is an IPv4 connection or if ADDRESS_CLASS is not TEMPORARY.
ON_LINK	ON_LINK	VARCHAR(3) Nullable	Whether or not this interface and all IPv6 addresses with the same prefix are on the same link. **NO** Addresses with the same prefix are not assumed to be on the same link. **YES** Addresses with the same prefix are assumed to be on the same link and directly reachable. Contains null if this is an IPv4 connection or if INTERNET_ADDRESS is *IP6SAC.
PROXY_ARP_ENABLED	PRXARPENB	VARCHAR(3) Nullable	Indicates whether Proxy ARP is currently active for this interface. **NO** Proxy ARP not enabled. **YES** Proxy ARP enabled. Contains null if this is an IPv6 connection.
PROXY_ARP_ALLOWED	PRXARPALW	VARCHAR(3) Nullable	Indicates whether Proxy ARP has been configured to be allowed or not allowed. **NO** Proxy ARP not allowed. **YES** Proxy ARP allowed. Contains null if this is an IPv6 connection or if interface is not Opticonnect (*OPC) or Virtual Ethernet.
CURRENT_PROXY_AGENT_LINE	PRXAGT	VARCHAR(10) Nullable	Name of the communication line description that is used with the IPv6 interface for virtual IP address (VIPA) proxy Neighbor Discovery. Contains null if this is an IPv4 connection or if no association has been made between this interface and another physical interface.
CURRENT_PROXY_AGENT_ LINE_VIRTUAL_LAN_ID	PRXAGTLAN	VARCHAR(4) Nullable	The virtual LAN to which the proxy agent line belongs according to IEEE standard 802.1Q. Can also contain the following special value: **NONE** There is no virtual LAN identifier associated with the current proxy agent line. Contains null if this is an IPv4 connection or if no association has been made between this interface and another physical interface.

Table 72. NETSTAT_INTERFACE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LAST_CHANGE_TIMESTAMP	LASTCHG	TIMESTAMP(0) Nullable	The timestamp of the most recent change to this interface in the dynamic tables used by the protocol stack. Contains null if the interface has never been changed.

Example

Return information about all interfaces which are using Virtual Ethernet protocol..

```
SELECT * FROM QSYS2.NETSTAT_INTERFACE_INFO
  WHERE INTERFACE_LINE_TYPE = 'VETH'
```

Related information

Internet Protocol version 6

NETSTAT_JOB_INFO view

The NETSTAT_JOB_INFO view returns information about jobs using IPv4 and IPv6 network connections.

The values returned for the columns in the view are closely related to the values returned by Retrieve Network Connection Data API. Refer to the API for more detailed information.

The following table describes the columns in the view. The system name is NS_JOB. The schema is QSYS2.

Table 73. NETSTAT_JOB_INFO view

Column Name	System Column Name	Data Type	Description
CONNECTION_TYPE	CONN_TYPE	CHAR(4)	The type of connection. **IPV4** The connection is an IPv4 connection. **IPV6** The connection is an IPv6 connection.
REMOTE_ADDRESS	RMT_ADDR	VARCHAR(45)	The internet address of the remote host. For IPv4: • The address is in IPv4 address format. A value of 0.0.0.0 indicates that either the system is waiting for a connection to open or that a UDP socket is being used. A value of 0 means that the connection is a listening or UDP socket so this field does not apply. For IPv6: • The address is in IPv6 address format. A value of :: means that the connection is a listening socket so this field does not apply.
REMOTE_PORT	RMT_PORT	INTEGER	The remote host port number. A value of 0 means that the connection is a listening or UDP socket, so this field does not apply.
REMOTE_PORT_NAME	RMT_NAME	VARGRAPHIC(14) CCSID 1200 Nullable	The remote host well-known port name or the name from the service table entry. Contains null if there is no well-known port name.
LOCAL_ADDRESS	LOCAL_ADDR	VARCHAR(45)	The local address of this connection on this system. For IPv4: • The address is in IPv4 address format. A value of 0.0.0.0 indicates that either the system is waiting for a connection to open or that a UDP socket is being used. For IPv6: • The address is in IPv6 address format. A value of :: means the local application specified that any local internet address can be used.
LOCAL_PORT	LOCAL_PORT	INTEGER	The local system port number.

Table 73. NETSTAT_JOB_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LOCAL_PORT_NAME	LOCAL_NAME	VARGRAPHIC(14) CCSID 1200 Nullable	The local system well-known port name or the name from the service table entry. Contains null if there is no well-known port name.
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10) Nullable	The effective user profile of the thread for which information is being retrieved. This name may differ from the user portion of the job name. Contains null when SLIC_TASK_NAME is not null or if JOB_NAME is the special value *SIGNON.
JOB_NAME	JOB_NAME	VARCHAR(28) Nullable	The qualified job name. Can also contain the following special value: ***SIGNON** This connection is a telnet connection and the system is performing sign-on processing or is displaying a sign-on prompt on it. Contains null when SLIC_TASK_NAME is not null.
SLIC_TASK_NAME	SLIC_TASK	VARCHAR(16) Nullable	The task name as identified to the system. Contains null when JOB_NAME is not null.
INTERNAL_JOB_ID	JOB_ID	BINARY(16) Nullable	A value that can be used by system APIs to speed the process of locating the job on the system. Contains null if JOB_NAME is the special value *SIGNON or if SLIC_TASK_NAME is not null.
JOB_TYPE	JOB_TYPE	VARCHAR(11) Nullable	The type of job: **AUTOSTART** The job is an autostart job. **BATCH** The job is a batch job. **INTERACTIVE** The job is an interactive job. **MONITOR** The job is a subsystem monitor job. **READER** The job is a spooled reader job. **SCPF** The job is the SCPF system job. **SYSTEM** The job is a system job. **WRITER** The job is a spooled writer job. Contains null if JOB_NAME is the special value *SIGNON or if SLIC_TASK_NAME is not null.

Example

Return information about all jobs using IPv4 network connections.

```
SELECT * FROM QSYS2.NETSTAT_JOB_INFO
  WHERE CONNECTION_TYPE = 'IPV4'
```

Related information

Internet Protocol version 6

NETSTAT_ROUTE_INFO view

The NETSTAT_ROUTE_INFO view returns information about IPv4 and IPv6 routes.

The values returned for the columns in the view are closely related to the values returned by List Network Routes API. Refer to the API for more detailed information.

The following table describes the columns in the view. The system name is NS_ROUTE. The schema is QSYS2.

Table 74. NETSTAT_ROUTE_INFO view

Column Name	System Column Name	Data Type	Description
CONNECTION_TYPE	CONN_TYPE	CHAR(4)	The type of connection. **IPV4** The connection is an IPv4 connection. **IPV6** The connection is an IPv6 connection.
ROUTE_DESTINATION	ROUTE_DEST	VARCHAR(45)	The Internet Protocol address of the ultimate destination reached by this route. For IPv4: • The address is in IPv4 address format. When used in combination with the subnet mask and the type of service values, the route destination identifies a route to a network or system. A value of 0.0.0.0 means that the route destination is the default route. For IPv6: • The address is in IPv6 address format. When used in combination with the prefix length, the route destination identifies a route to a network or host.
SUBNET_MASK	SUBNET_MSK	VARCHAR(15) Nullable	The actual value of the subnet mask for the route destination in dotted-decimal notation. A value of 0.0.0.0 means no value is defined. Contains null if this is an IPv6 connection.
NEXT_HOP	NEXT_HOP	VARCHAR(45)	The internet address of the first system on the path from your system to the route destination. For IPv4: • The address is in IPv4 address format. For IPv6: • The address is in IPv6 address format. Can contain the following special value: ***DIRECT** This is the next hop value of a route that is automatically created.
PREFIX_LENGTH	PRE_LEN	INTEGER Nullable	The prefix length defines how many bits of the route destination IPv6 address are in the prefix. It specifies how many of the left-most bits of the address make up the prefix. The prefix length is used to generate network and host addresses. Contains null if this is an IPv4 connection.
ROUTE_STATUS	ROUTE_STS	VARCHAR(10) Nullable	The current state of the route. **DOD** This route is used for Point-to-Point (PPP) Dial-on-Demand. Currently, this Dial-on-Demand route is not available. The route will become available when a Dial-on-Demand session is initiated for the interface this route is associated with. For IPv4: **YES** The router specified by the next hop value for this interface is available for use. **NO** The router specified by the next hop value for this interface is not available for use. **NO GATEWAY** The router specified by the next hop value for this interface is not available for use, the router may be experiencing a problem. For IPv6: **ACTIVE** This route is currently active and is in the current route search path. **INACTIVE** This route is not in the route search path and is not being used. Contains null if the state is unknown.

Table 74. NETSTAT_ROUTE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
ROUTE_MAXIMUM_ TRANSMISSION_UNIT	ROUTE_MTU	VARCHAR(10)	The maximum transmission unit (MTU) value for this route in bytes. Can be either a number or one of the following special values: For IPv4: **IFC** The route is not currently active and the MTU was specified as *IFC. **OTHER** An Internet Protocol (IP) over SNA interface. For IPv6: ***IP6LINMTU** This route uses the MTU of the line it is bound to.
CONFIGURED_ROUTE_ MAXIMUM_ TRANSMISSION_UNIT	CFG_RT_MTU	VARCHAR(10) Nullable	A number representing the configured maximum transmission unit (MTU) value for this route, in bytes. Can be either a number or the following special value: ***IP6LINMTU** The route MTU was specified as *IP6LINMTU, the MTU value of the line to which this route is bound. Contains null if this is an IPv4 connection.
ROUTE_TYPE	ROUTE_TYPE	VARCHAR(8) Nullable	The type of route. **DFTROUTE** A default route. **DIRECT** A route to a network or subnetwork to which this system has a direct physical connection. **HOST** A route to a specific remote host. **NET** An indirect route to a remote network. **SUBNET** An indirect route to a remote subnetwork. This option is only for IPv4 connections. Contains null if the type of route is unknown.

Table 74. NETSTAT_ROUTE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
ROUTE_SOURCE	ROUTE_SRC	VARCHAR(18) Nullable	Specifies how this route was added to the routing table. For IPv4: **CFG** The route was added using the configuration commands of the local system. **ICMP** The route was added with the Internet Control Message Protocol (ICMP) redirect mechanism. **OTHER** The route was added with a sockets input/output control (IOCtl) or other mechanism. **RIP** The route was added by the Routing Information Protocol (RIP). **SNMP** The route was added by the Simple Network Management Protocol (SNMP). For IPv6: **AUTOCONFIG** This route was added because of an interface added by stateless autoconfiguration. **BGP** This route was added by the Border Gateway Protocol (BGP). **CFGIFC** The route was added because of a manually configured interface. **CFGRTE** The route was manually configured. **IDRP** This route was added by the Inter-Domain Routing Protocol (IDRP). **IGRP** This route was added by the Interior Gateway Routing Protocol (IGRP). **OSPF** The route was added by the Open Shortest Path First (OSPF) protocol. **RA_PREFIX_INFO** This route was added because of the presence of a Prefix Information Option on a Router Advertisement packet received by the system. **RA_ROUTE_INFO** This route was added because of the presence of a Route Information Option on a Router Advertisement packet received by the system. **RA_ROUTER_LIFETIME** This route was added because of the presence of a non-zero value in the Router Lifetime field in a Router Advertisement packet received by the system. **REDIRECT** This route was added by the ICMPv6 redirect mechanism. **RIP** The route was added by the Routing Information Protocol (RIP). **ROUTING** This route was determined to be necessary and added by the TCP/IP stack on this system. **SNMP** This route was added by the Simple Network Management Protocol (SNMP). Contains null if the route source is not known.

Table 74. NETSTAT_ROUTE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
SERVICE_TYPE	SRVC_TYPE	VARCHAR(9) Nullable	The type of service that defines how the internet hosts and routers should make trade-offs between throughput, delay, reliability, and cost. **MAXRLB** A higher level of effort to ensure delivery is important for datagrams with the maximize reliability indication. **MAXTHRPUT** High data rate is important for datagrams with the maximize throughput indication. **MINCOST** Lower cost is important for datagrams with the minimize monetary cost indication. **MINDELAY** Prompt delivery is important for datagrams with the minimize delay indication. **NORMAL** Normal service is used for delivery of datagrams. **OTHER** An Internet Protocol (IP) over SNA interface. Contains null if this is an IPv6 connection.
ROUTE_PROTOCOL	ROUTE_PTCL	VARCHAR(7) Nullable	Specifies the protocol that was used to generate this route. **BGP** Border Gateway protocol. **IDRP** InterDomain Routing protocol. **IGRP** InterGateway Routing protocol. **LOCAL** Local configuration. **NDISC** Neighbor discovery. **NETMGMT** Network Management protocol. **OSPF** Open Shortest Path First protocol. **OTHER** None of the listed protocols. **RIP** Routing Information protocol. Contains null if this is an IPv4 connection.
ROUTE_PREFERENCE	ROUTE_PREF	VARCHAR(6) Nullable	The preference of this route during route selection. **LOW** This route has a low preference. **MEDIUM** This route has a medium preference. **HIGH** This route has a high preference. Contains null if this is an IPv4 connection.
LOCAL_BINDING_TYPE	LOCALTYPE	VARCHAR(7) Nullable	The type of line to which this route is bound. • **DYNAMIC** • **STATIC** Contains null if this is an IPv6 connection.
LOCAL_BINDING_INTERFACE	LOCALIFC	VARCHAR(15) Nullable	The IP interface to bind to this route. Contains null if this is an IPv6 connection.

Table 74. NETSTAT_ROUTE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LOCAL_BINDING_INTERFACE_ STATUS	LOCALSTS	VARCHAR(12) Nullable	The current status of the logical interface. **ACTIVE** The interface has been started and is running. **DOD** This interface is being used for Point-to-Point (PPP) Dial-on-Demand. **DUPLICATE** Another host on the LAN responded to a packet destined for this logical interface. **ENDING** The operating system is processing the request to end this interface. **FAILED** The line description associated with this interface has entered the failed state. **FAILED (TCP)** An error was detected in the IBM TCP/IP Licensed Internal Code. **INACTIVE** The interface has not been started. **RCYCNL** A hardware failure has occurred and the line description associated with this interface is in the recovery canceled (RCYCNL) state. **RCYPND** An error with the physical line associated with this interface was detected by the system. The line description associated with this interface is in the recovery pending (RCYPND) state. **STARTING** The operating system is processing the request to start this interface. Contains null if this is an IPv6 connection.
LOCAL_BINDING_NETWORK_ ADDRESS	LOCALADDR	VARCHAR(15) Nullable	The Internet address, in dotted decimal notation, of the IP network or subnetwork that the interface is attached to. Contains null if this is an IPv6 connection.
LOCAL_BINDING_SUBNET_MASK	LOCALMASK	VARCHAR(15) Nullable	The subnet mask for the network, subnet, and host address fields for the local binding network address, in dotted decimal notation, that defines the subnetwork for an interface. Contains null if this is an IPv6 connection.
LOCAL_BINDING_LINE_ DESCRIPTION	LOCALLINE	VARCHAR(10)	The name of the communications line description or virtual line (L2TP) that identifies the network associated with an interface. Can contain the following special values: ***LOOPBACK** This is a loopback interface. Processing associated with the loopback interface does not extend to a physical line. ***OPC** This interface is attached to the optical bus (OptiConnect). ***VIRTUALIP** The virtual interface is a circuitless interface.
LOCAL_BINDING_LINE_STATUS	LOCALLSTS	VARCHAR(8) Nullable	The current operational status of the communications line to which this route is bound. **ACTIVE** The line is operational. **FAILED** The desired state of the line is Active, but it is currently in the Inactive state. **INACTIVE** The line is not operational. Contains null if this is an IPv4 connection.

Table 74. NETSTAT_ROUTE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LOCAL_BINDING_LINE_TYPE	LOCALLTYPE	VARCHAR(6)	The type of line used by the interface. **ASYNC** Asynchronous communications protocol. **DDI** Distributed Data Interface protocol. **ELAN** Ethernet local area network protocol. **FR** Frame relay network protocol. **L2TP** Layer Two Tunneling protocol. **PPP** Point-to-Point protocol. **PPPOE** Point-to-Point over Ethernet protocol. **TDLC** Twinaxial Datalink Control. Used for TCP/IP over Twinax. **TRLAN** Token-ring local area network protocol. **VETH** Virtual Ethernet protocol. **WLS** Wireless local area network protocol. **X25** X.25 protocol. Can also contain one of the following special values: **ERROR** A system error other than those for NOTFND was received while trying to determine the link type for an interface. **NONE** Line is not defined. This value is used for the following interfaces: *LOOPBACK, *VIRTUALIP, *OPC. There is no line type value for this interface. **NOTFND** Not found. The line description object for this interface cannot be found. **OTHER** An Internet Protocol (IP) over SNA interface.
LOCAL_BINDING_VIRTUAL_ LAN_ID	LOCALLAN	VARCHAR(4)	The virtual LAN to which this route is bound. Can also contain the following special value: **NONE** No virtual LAN identifier is associated with the binding line.
ROUTE_PRECEDENCE	ROUTE_PRCD	INTEGER Nullable	Priority of route. Values are 1 to 10, with the lowest priority being 1. Contains null if this is an IPv6 connection.
ROUTE_TEXT	LABEL	VARGRAPHIC(50) CCSID(1200) Nullable	Text description associated with the route. Contains null if there is no description.
DUPLICATE	DUPLICATE	VARCHAR(6) Nullable	Indicates whether this route is a duplicate of another route in the routing table or not, and also whether there are any routes which are duplicates of this route. **NO** This route is not a duplicate of another route but it does have duplicates. **UNIQUE** This route is not a duplicate of another route and it does not have any duplicates. **YES** This route is a duplicate of another route. Contains null if this is an IPv4 connection.

Table 74. NETSTAT_ROUTE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
EXPIRATION	EXPIRATION	TIMESTAMP(0) Nullable	The timestamp when this route will expire or did expire. If the timestamp is in the future, the route has not expired yet. If the timestamp is in the past, then this route has expired and is still being returned for a short period of time to indicate that the route ceased to function because its lifetime expired. Contains null if this is an IPv4 connection or if the route will never expire.
PPP_CONFIGURATION_PROFILE	PPPCFGPRF	VARCHAR(10) Nullable	The name of the Point-to-Point Protocol (PPP) configuration profile associated with this route. Contains null if this is an IPv4 connection or if Point-to-Point Protocol is not being used with this route.
PPP_AUTHENTICATION_USER_ID	PPPAUTUSR	VARCHAR(24) Nullable	The Point-to-Point Protocol authentication user id associated with this route. Contains null if this is an IPv4 connection or if Point-to-Point Protocol is not being used with this route.
PPP_INTERNET_ADDRESS	PPPINTADD	VARCHAR(45) Nullable	The internet address, in IPv6 address format, to which this Point-to-Point route is bound. Contains null if this is an IPv4 connection or if Point-to-Point Protocol is not being used with this route.
PPP_DIAL_ON_DEMAND_ PROFILE	PPPDODPRF	VARCHAR(10) Nullable	The name of the Dial-on-demand Remote Peer Enabled Point-to-Point profile associated with this route. Contains null if this is an IPv4 connection or if Point-to-Point Protocol is not being used with this route.
LAST_CHANGE_TIMESTAMP	LASTCHG	TIMESTAMP(0) Nullable	The timestamp of the most recent change to this route in the dynamic tables used by the protocol stack. Contains null if the interface has never been changed.

Example

Return information about all routes which are available for use.

```
SELECT * FROM QSYS2.NETSTAT_ROUTE_INFO
  WHERE ROUTE_STATUS = 'YES' OR ROUTE_STATUS = 'ACTIVE'
```

Related information

Internet Protocol version 6

SET_SERVER_SBS_ROUTING procedure

The SET_SERVER_SBS_ROUTING procedure provides the ability to configure some servers to utilize alternative subsystems based on the user profile that is establishing the connection. The user profile can be a group profile or a supplemental group profile.

This procedure allows an administrator to reposition specific users into alternate, non-default, subsystems. When configured, new incoming TCP/IP server connections will use the alternate subsystem. If the server is configured to route to a user-specified subsystem by incoming TCP/IP address as well, the job will first route to the subsystem configured for that TCP/IP address and then immediately attempt to route to the subsystem configured for the connecting user profile.

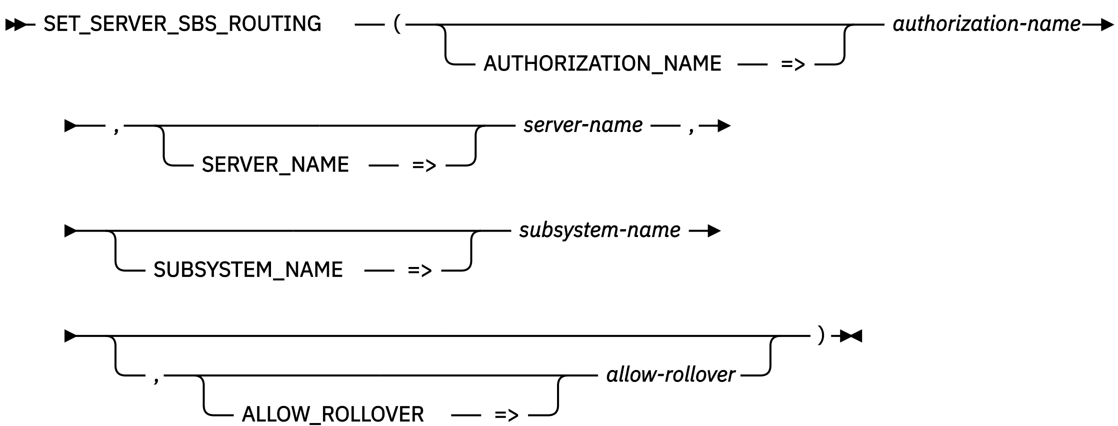
By default, all users for the following servers utilize the same subsystem:

Table 75. Servers and default subsystems

Server Description	Server Name	Default subsystem
Central server	QZSCSRVS	QUSRWRK
Database server	QZDASOINIT	QUSRWRK
Data queue server	QZHQSSRV	QUSRWRK
DDM	QRWTSRVR	QUSRWRK

Table 75. Servers and default subsystems (continued)		
Server Description	**Server Name**	**Default subsystem**
DRDA	QRWTSRVR	QUSRWRK
File server	QPWFSERVSO	QSERVER
Network print server	QNPSERVS	QUSRWRK
Remote command server	QZRCSRVS	QUSRWRK

For more information on these servers see DRDA and DDM overview and Host servers by function.



The schema is QSYS2.

authorization-name
A character or graphic string expression that identifies an existing user or group profile name.

server-name
A character or graphic string expression that identifies the name of the server job that will be rerouted to *subsystem-name* for *authorization-name* whenever a connection is initiated to this server job. Valid *server-name* values are:

- QNPSERVS
- QPWFSERVSO
- QRWTSRVR
- QZDASOINIT
- QZHQSSRV
- QZRCSRVS
- QZSCSRVS

The special value of *ALL can be used to indicate all of the valid *server-name* values.

subsystem-name
A character or graphic string expression that identifies the name of the subsystem that will be used for the specified user, instead of the default subsystem, whenever a connection is initiated to the specified server job. No validation is done on *subsystem-name* to make sure it is a valid and active subsystem.

Specifying the null value will clear the entry for this *authorization-name* and *server-name*.

allow-rollover
A character or graphic string expression that indicates the action to take if the specified subsystem is not active. Valid values are:

NO
If the alternate subsystem cannot be used, the connection request will fail.

YES
If the alternate subsystem cannot be used, the connection request will succeed by using a batch immediate job in the default subsystem.

If this parameter is not specified, the default is YES.

Notes

Authorization: The user calling this procedure must have *SECADM special authority. In addition, *OBJMGT and *USE is required to the user profile.

The Prestart Job Entry must specify the *subsystem-name*.

If, for any reason, the alternate subsystem cannot be used to establish the connection, the connection will run in the default subsystem (or the last subsystem it was successfully routed to) as a batch immediate job. An example of this would be if the *authorization-name* does not have *USE authority to the subsystem description for *subsystem-name*.

If routing has been configured for a user profile, the user profile configuration will always be used, regardless of any group profile configuration. A group profile configuration will take precedence over any supplemental group profile configuration.

Examples

- Set new incoming DRDA and DDM TCP/IP server connections for user profile TIM to route to subsystem TIMSUBSYS.

```
CALL QSYS2.SET_SERVER_SBS_ROUTING('TIM','QRWTSRVR','TIMSUBSYS')
```

- Reset incoming DRDA and DDM TCP/IP server connections for user profile TIM back to the original default subsystem.

```
CALL QSYS2.SET_SERVER_SBS_ROUTING('TIM','QRWTSRVR',NULL)
```

- Configure group profile ADMIN to use an alternate subsystem for all of the servers supported by this procedure. Do not permit a connection request to rollover to use QUSRWRK.

```
CALL QSYS2.SET_SERVER_SBS_ROUTING('ADMIN','*ALL','ADHOCSBS','NO')
```

- Set new incoming Database server TCP/IP connections for user profile BOB to route to subsystem BOBSUBSYS.

```
CALL QSYS2.SET_SERVER_SBS_ROUTING('BOB','QZDASOINIT','BOBSUBSYS')
```

- Construct a subsystem that will constrain the amount of system resources available to users who are known to execute expensive queries.

```
CRTSBSD SBSD(QGPL/ADHOCSBS) POOLS((1 *BASE)) TEXT('Adhoc DRDA users SBS')
CRTJOBQ QGPL/ADHOCJOBQ TEXT('Adhoc DRDA users job queue')
ADDJOBQE SBSD(QGPL/ADHOCSBS) JOBQ(QGPL/ADHOCJOBQ) MAXACT(25) SEQNBR(40)
CRTCLS CLS(QGPL/ADHOCCLS) RUNPTY(55) TIMESLICE(100) TEXT('Adhoc DRDA users class')
ADDPJE SBSD(QGPL/ADHOCSBS) PGM(QSYS/QRWTSRVR) JOBD(QGPL/QDFTSVR) CLS(QGPL/ADHOCCLS)
STRSBS SBSD(QGPL/ADHOCSBS)
CALL QSYS2.SET_SERVER_SBS_ROUTING('SLFUSER','QRWTSRVR','ADHOCSBS')
```

Related information
Use of prestart jobs

SERVER_SBS_ROUTING view

The SERVER_SBS_ROUTING view returns information about the users who have alternate subsystem configurations for some IBM i servers. When a user profile listed in this view attempts to use TCP/IP to form a connection to the server, an attempt is made to use the alternate subsystem instead of the default subsystem for that server.

Authorization: You must have *OBJOPR and *READ authority to a *USRPRF or it will not be returned.

The QSYS2.SET_SERVER_SBS_ROUTING procedure can be used to change the values shown in this view.

The following table describes the columns in the view. The system name is SRVR_RTG. The schema is QSYS2.

Table 76. SERVER_SBS_ROUTING view

Column Name	System Column Name	Data Type	Description
AUTHORIZATION_NAME	USER_NAME	VARCHAR(128)	The user profile that has an alternate subsystem configuration.
QRWTSRVR_SUBSYSTEM	DRDADDMSBS	VARCHAR(10) Nullable	The subsystem name that incoming DRDA or DDM connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QZDASOINIT_SUBSYSTEM	ZDASBS	VARCHAR(10) Nullable	The subsystem name that incoming database server connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QZRCSRVS_SUBSYSTEM	ZRCSBS	VARCHAR(10) Nullable	The subsystem name that incoming remote command server connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QZHQSSRV_SUBSYSTEM	ZHQSBS	VARCHAR(10) Nullable	The subsystem name that incoming data queue server connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QZSCSRVS_SUBSYSTEM	ZSCSBS	VARCHAR(10) Nullable	The subsystem name that incoming central server connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QNPSERVS_SUBSYSTEM	NPSSBS	VARCHAR(10) Nullable	The subsystem name that incoming network print server connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QPWFSERVSO_SUBSYSTEM	PWFSBS	VARCHAR(10) Nullable	The subsystem name that incoming file server connections will be rerouted to. Contains the null value when an alternate subsystem for this server has not been configured for this user.
QRWTSRVR_ROLLOVER	DRDA_RO	VARCHAR(3)	Indicates how incoming DRDA or DDM connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.
QZDASOINIT_ROLLOVER	ZDA_RO	VARCHAR(3)	Indicates how incoming database server connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.

Table 76. SERVER_SBS_ROUTING view (continued)

Column Name	System Column Name	Data Type	Description
QZRCSRVS_ROLLOVER	ZRC_RO	VARCHAR(3)	Indicates how incoming remote command connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.
QZHQSSRV_ROLLOVER	ZHQ_RO	VARCHAR(3)	Indicates how incoming data queue server connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.
QZSCSRVS_ROLLOVER	ZSC_RO	VARCHAR(3)	Indicates how incoming central server connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.
QNPSERVS_ROLLOVER	NPS_RO	VARCHAR(3)	Indicates how incoming network print server connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.
QPWFSERVSO_ROLLOVER	PWF_RO	VARCHAR(3)	Indicates how incoming file server connection requests are handled if the specified subsystem cannot be used. **NO** Incoming connections will be rejected. **YES** Incoming connections will be routed to the default subsystem. If an alternate subsystem is not configured for this server, YES is the default.

The following table shows the servers that can have alternate subsystem configurations.

Table 77. Servers and server names

Server Description	Server Name
Central server	QZSCSRVS
Database server	QZDASOINIT
Data queue server	QZHQSSRV
DDM	QRWTSRVR
DRDA	QRWTSRVR
File server	QPWFSERVSO
Network print server	QNPSERVS
Remote command server	QZRCSRVS

Example

Query subsystem routing information for all user profiles:

```
SELECT * FROM QSYS2.SERVER_SBS_ROUTING
```

TCPIP_INFO view

The TCPIP_INFO view contains TCP/IP information for the current host connection.

The following table describes the columns in the view. The schema is QSYS2.

Table 78. TCPIP_INFO view

Column Name	System Column Name	Data Type	Description
COLLECTED_TIME	COLLE00001	TIMESTAMP Nullable	Timestamp indicating when this row of information was collected.
LOCAL_HOST_NAME	LOCAL00001	VARCHAR(255) Nullable	TCP/IP host name of the local system.
CLIENT_IP_ADDRESS_TYPE	CLIEN00001	VARCHAR(10) Nullable	TCP/IP address version of the client.
CLIENT_IP_ADDRESS	CLIEN00002	VARCHAR(45) Nullable	TCP/IP address of the client.
CLIENT_PORT_NUMBER	CLIEN00003	INTEGER Nullable	TCP/IP port of the client.
SERVER_IP_ADDRESS_TYPE	SERVE00001	VARCHAR(10) Nullable	TCP/IP address version of the server.
SERVER_IP_ADDRESS	SERVE00002	VARCHAR(45) Nullable	TCP/IP address of the server.
SERVER_PORT_NUMBER	SERVE00003	INTEGER Nullable	TCP/IP port number of the server.
HOST_VERSION	HOST_00001	VARCHAR(10) Nullable	Operating system version.

Example

Return information about the current host connection.

```
SELECT * FROM QSYS2.TCPIP_INFO
```

Java Services

This view and procedure provide Java information and JVM management options.

JVM_INFO view

The JVM_INFO view returns information about active Java Virtual Machine (JVM) jobs. The information is a subset of what can be found interactively using the Work with JVM Jobs (WRKJVMJOB) command.

The following table describes the columns in the view. The schema is QSYS2.

Table 79. JVM_INFO view

Column Name	System Column Name	Data Type	Description
JOB_NAME	JOB_NAME	VARCHAR(28)	The qualified job name for the active JVM.

Table 79. JVM_INFO view (continued)

Column Name	System Column Name	Data Type	Description
PROCESS_ID	PROCESS_ID	INTEGER	The process identifier used by the kernel to uniquely identify the process.
START_TIME	START_TIME	TIMESTAMP	The current time when the JVM was started.
INITIAL_THREAD_TASKCOUNT	INITTHDNUM	BIGINT	The number of threads initiated for this JVM when the JVM was started.
JAVA_THREAD_COUNT	JAVATHDNUM	BIGINT	The current number of java threads within the JVM job.
TOTAL_GC_TIME	ACCUMTIME	BIGINT	Total time spent performing garbage collection tasks in milliseconds.
GC_CYCLE_NUMBER	GC_CYCLE	INTEGER	The current or last garbage collection cycle performed.
GC_POLICY_NAME	GCPOLICY	VARGRAPHIC(16) CCSID 1200	The name of the garbage collection policy in use.
JAVA_HOME	JAVA_HOME	VARGRAPHIC(1024) CCSID 1200	The java.home environment variable value in effect for this JVM. This value indicates the JDK that is used when running a Java application. The location of the Java tools and utilities is in one of two directories, either <JAVA_HOME>/jre/bin or <JAVA_HOME>/bin, where <JAVA_HOME> is the value of the JAVA_HOME environment variable. For example, if JAVA_HOME is set to /QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit, indicating that IBM Technology for Java 6 32-bit is to be used, then the Java tools and utilities directories would be: /QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit/bin /QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit/jre/bin
USER_DIRECTORY	USER_DIR	VARGRAPHIC(1024) CCSID 1200	The user working directory for the JVM. This also indicates the location where diagnostic detail will be dumped for the JVM.
NUM_CURRENT_PROPERTIES	NUMPROP	INTEGER	Total number of Java system properties currently present.
INITIAL_HEAP_SIZE	INTHEAP	BIGINT	The initial heap size available to the JVM code, in kilobytes.
CURRENT_HEAP_SIZE	CURHEAP	BIGINT	The amount of memory, in kilobytes, currently allocated for heap space.
IN_USE_HEAP_SIZE	INUSEHEAP	BIGINT	The amount of memory, in kilobytes, currently in use by the heap.
MAX_HEAP_SIZE	MAXHEAP	BIGINT	The maximum heap size available to the JVM code, in kilobytes.
MALLOC_MEMORY_SIZE	MALLOCSIZE	BIGINT	The amount of memory, in kilobytes, that has been allocated with malloc().
INTERNAL_MEMORY_SIZE	INTMEM	BIGINT	The amount of memory, in kilobytes, that the JVM is using for internal operations.
JIT_MEMORY_SIZE	JITSIZE	BIGINT	The size of the memory space, in kilobytes, that is used by the JIT (Just in Time) compiler.
SHARED_CLASS_SIZE	SHAREDSIZE	BIGINT	The amount of memory, in kilobytes, that the JVM is using for shared classes.
BIT_MODE	BIT_MODE	INTEGER	The Java version of this job. **32** 32 bit Java job **64** 64 bit Java job

Example

Examine the active JVM jobs, ordered by top heap space consumption.

```
SELECT * FROM QSYS2.JVM_INFO
   ORDER BY CURRENT_HEAP_SIZE DESC
```

SET_JVM procedure

The SET_JVM procedure can be used to manage specific JVM jobs.

This actions provided by this DB2 for i procedure can also be accomplished interactively using the Work with JVM Jobs (WRKJVMJOB) command.

►► SET_JVM — (— *job_name* — , — *action* —) ►◄

The schema is QSYS2.

job_name
A character or graphic string expression that identifies the qualified job name of the job to change.

action
A character or graphic string expression that specifies that action to perform. Supported actions are:

GC_ENABLE_VERBOSE
Enable verbose garbage collection detail.

GC_DISABLE_VERBOSE
Disable verbose garbage collection detail.

GENERATE_HEAP_DUMP
Generates information about the JVM's heap. Generates a dump of all the heap space allocations which have not yet been freed.

GENERATE_SYSTEM_DUMP
Generates system detail for the JVM. Generates a binary format raw memory image of the job that was running when the dump was initiated.

GENERATE_JAVA_DUMP
Generates Java detail for the JVM. Generates multiple files that contain diagnostic information for the JVM and the Java applications running within the JVM.

Example

- Change a specific web admin JVM to provide verbose garbage collection details:

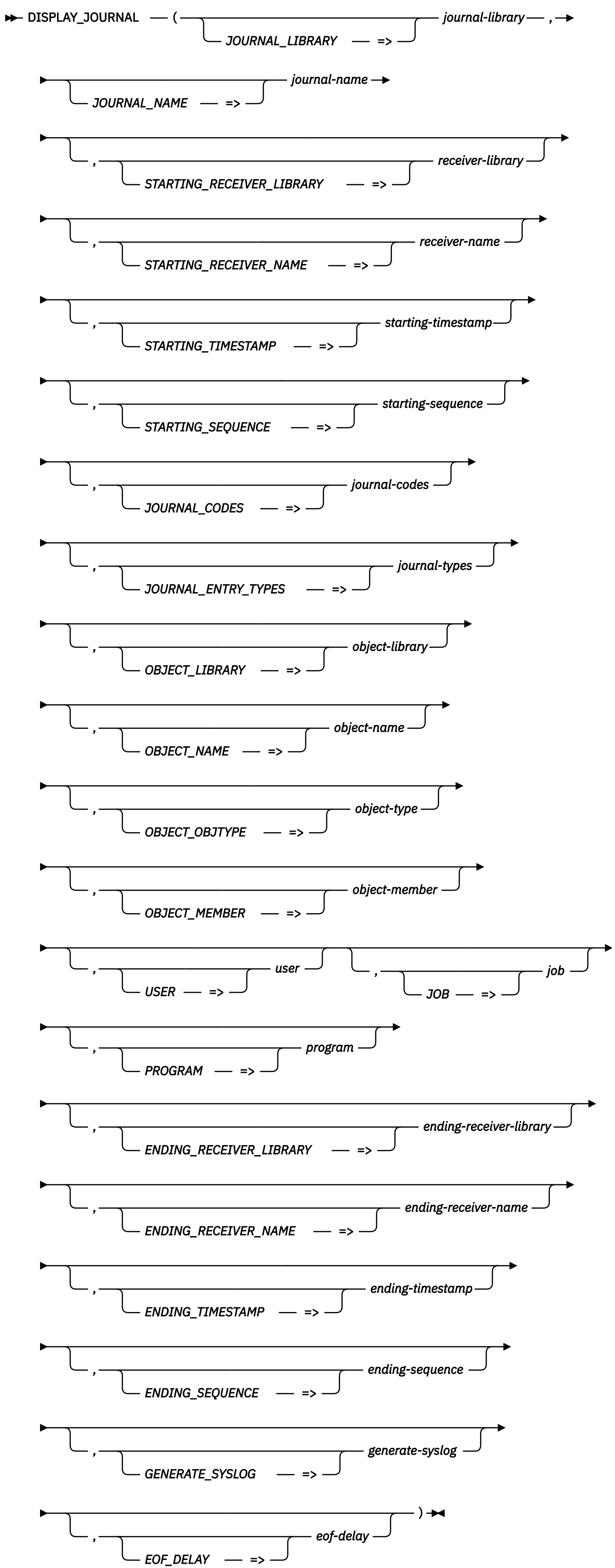
```
CALL QSYS2.SET_JVM('121376/QWEBADMIN/ADMIN4','GC_ENABLE_VERBOSE') ;
```

Journal Services

This function and view provide journal information.

DISPLAY_JOURNAL table function

The DISPLAY_JOURNAL table function returns information about journal entries. It returns information similar to what is returned by the Display Journal (DSPJRN) CL command and the Retrieve Journal Entries (QjoRetrieveJournalEntries) API.



The schema is QSYS2.

Authorization:

- You must have *USE authority to the journal and to all requested journal receivers.
- *OBJEXIST authority is required to the journal if *object-name* is omitted or if *object-name* specifies an object that no longer exists.
- If *object-name* is *ALL, you must be authorized to every object associated with a journal entry.

journal-library
A character or graphic string expression that identifies the name of the library containing the journal. The name cannot be *LIBL or *CURLIB.

journal-name
A character or graphic string expression that identifies the name of the journal.

receiver-library
A character or graphic string expression that identifies the name of the starting journal receiver library. The name can be *LIBL or *CURLIB.

receiver-name
A character or graphic string expression that identifies the name of the starting journal receiver. If one of the special values *CURRENT, *CURCHAIN, and *CURAVLCHN is specified, the *receiver-library* value will be ignored. Otherwise, the *receiver-name* and *receiver-library* must identify a valid journal receiver.

If no journal receiver is specified, *CURRENT will be used.

starting-timestamp
A timestamp value that specifies the starting timestamp to use.

A value cannot be specified for both *starting-timestamp* and *starting-sequence*.

starting-sequence
A decimal expression that identifies the starting sequence number to use. If the *starting-sequence* value is not found in the receiver range, an error is returned.

A value cannot be specified for both *starting-timestamp* and *starting-sequence*.

journal-codes
A character or graphic string expression that lists the journal codes to return. The string can contain the special values of *ALL or *CTL, or it can be a list of one or more journal codes. Journal codes in the string can be separated by one or more separators. Separators are blank and comma. For example, a valid string can be 'RJ' or 'R J' or 'R,J' or 'R, J'.

If no string is provided, *ALL is used.

journal-types
A character or graphic string expression that lists the journal entry types to return. The string can contain the special values of *ALL or *RCD, or it can be a list of one or more journal entry types. Journal entry types in the string can be separated by one or more separators. Separators are blank and comma. For example, a valid string can be 'JFCT' or 'JF CT' or 'JF,CT' or 'JF, CT'.

If no string is provided, *ALL is used.

object-library
A character or graphic string expression that identifies the name of an object library. The values *LIBL and *CURLIB are allowed.

object-name
A character or graphic string expression that identifies the name of an object.

If the *object-name* contains the special value of *ALL, *object-library* must contain a library name and *object-type* must contain a valid object type. Otherwise, *object-name*, *object-library*, *object-type*, and *object-member* must identify a valid object.

object-type
A character or graphic string expression that identifies the system object type for the object. The value must be *DTAARA, *DTAQ, *FILE, or *LIB.

object-member
A character or graphic string expression that identifies the name of a member. It can be a special value of *FIRST, *ALL, or *NONE or a valid member name. If the object type is not *FILE, the member name is ignored.

user
A character or graphic string expression that identifies the user profile name for the current user of the job. If *user* is not specified, *ALL is used.

job
A character or graphic string expression that identifies the name of a job. The first 10 characters are the job name, the second 10 characters are the user name, and the last 6 characters are the job number. If *job* is not specified, *ALL is used.

program
A character or graphic string expression that identifies the name of a program. If *program* is not specified, *ALL is used.

ending-receiver-library
A character or graphic string expression that identifies the name of the ending journal receiver library. The name can be *LIBL or *CURLIB. If *ending-receiver-name* is not *CURRENT, a value for *ending-receiver-library* must be specified.

The value of this parameter is ignored if *eof-delay* is greater than zero.

ending-receiver-name
A character or graphic string expression that identifies the name of the ending journal receiver. If the special value *CURRENT is specified, the *ending-receiver-library* value will be ignored. Otherwise, the *ending-receiver-name* and *ending-receiver-library* must identify a valid journal receiver.

If *ending-receiver-name* is not specified, *CURRENT is used.

The value of this parameter is ignored if *eof-delay* is greater than zero.

ending-timestamp
A timestamp value that specifies the ending timestamp to use.

A value cannot be specified for both *ending-timestamp* and *ending-sequence*. This parameter cannot be specified if *eof-delay* is greater than zero.

ending-sequence
A decimal expression that identifies the ending sequence number to use. If the *ending-sequence* value is not found in the receiver range, an error is returned.

A value cannot be specified for both *ending-timestamp* and *ending-sequence*. This parameter cannot be specified if *eof-delay* is greater than zero.

generate-syslog
A character or graphic string expression that indicates whether to transform journal entries into syslog formatted detail. Values are:

NO
No syslog information will be returned. The SYSLOG_EVENT, SYSLOG_FACILITY, SYSLOG_SEVERITY, and SYSLOG_PRIORITY columns will contain the null value.

RFC3164
Values will be returned for the SYSLOG_EVENT, SYSLOG_FACILITY, SYSLOG_SEVERITY, and SYSLOG_PRIORITY columns if syslog information is defined for the journal entry. The SYSLOG_EVENT column will contain a syslog header that matches the RFC3164 format as described by the Internet Engineering Task Force (IETF) Request For Comments (RFC) 3164.

RFC5424

Values will be returned for the SYSLOG_EVENT, SYSLOG_FACILITY, SYSLOG_SEVERITY, and SYSLOG_PRIORITY columns if syslog information is defined for the journal entry. The SYSLOG_EVENT column will contain a syslog header that matches the RFC5424 format as described by the Internet Engineering Task Force (IETF) Request For Comments (RFC) 5424.

DISPLAY_JOURNAL only returns syslog information for the audit journal. If RFC3164 or RFC5424 is specified, *journal-library* must be QSYS and *journal-name* must be QAUDJRN.

If *generate-syslog* is not specified or is the null value, NO is used.

eof-delay

An integer expression that specifies the number of seconds to sleep when all audit journal entries have been read. This delay allows the caller to establish a polling service that will continually return rows, sleeping for the specified interval whenever all entries have been processed.

A value of zero indicates no delay is used and a finite set of rows will be returned. A value greater than zero indicates that the table function will sleep, as needed, to wait for new audit journal entries and never end. If *eof-delay* is not specified or is the null value, zero is used.

If this parameter has a value greater than zero, the *generate-syslog* parameter must be RFC3164 or RFC5424, the *ending-receiver-library* and *ending-receiver-name* are ignored, and the *ending-timestamp* and *ending-sequence* parameters cannot be specified with a value other than their default values.

When using a non-zero *eof-delay* parameter, avoid using query clauses that depend on returning a finite number of rows. For example, using the FETCH FIRST n ROWS clause can cause the query to end when the requested number of rows has been satisfied. A query using the DISPLAY_JOURNAL function with a non-zero *eof-delay* parameter does not allow data to be copied (ALWCPYDTA(*NO)). This means that a query requiring a copy of data, such as one using an ORDER BY clause or UNION DISTINCT, will issue an error and not be allowed.

The special values supported for the function arguments are the same as for the Display Journal (DSPJRN) CL command.

The result of the function is a table containing rows with the format shown in the following table. All the columns are nullable.

Table 80. DISPLAY_JOURNAL table function

Column Name	Data Type	Description
ENTRY_TIMESTAMP	TIMESTAMP	The system date and time when the journal entry was added to the journal receiver.
SEQUENCE_NUMBER	DECIMAL(21,0)	A number assigned by the system to each journal entry.
JOURNAL_CODE	CHAR(1)	The primary category of the journal entry.
JOURNAL_ENTRY_TYPE	CHAR(2)	Further identifies the type of user-created or system-created entry.
COUNT_OR_RRN	BIGINT	Contains either the relative record number (RRN) of the record that caused the journal entry or a count that is pertinent to the specific type of journal entry.
ENTRY_DATA	BLOB(2G)	The entry specific data returned for this journal entry.
NULL_VALUE_INDICATORS	VARCHAR(8000)	The null value indicators returned for this journal entry.
OBJECT	VARCHAR(30)	The name of the object for which the journal entry was added.
OBJECT_TYPE	VARCHAR(10)	The type of object in the entry.
OBJECT_TYPE_INDICATOR	CHAR(1)	An indicator with respect to the information in the object field.
FILE_TYPE_INDICATOR	CHAR(1)	Identifies whether or not this journal entry is associated with a logical file.
JOURNAL_IDENTIFIER	VARCHAR(10)	The journal identifier (JID) for the object.
CURRENT_USER	VARCHAR(10)	The name of the effective user profile under which the job was running when the entry was created.
JOB_NAME	VARCHAR(10)	The name of the job that added the entry.

Table 80. DISPLAY_JOURNAL table function (continued)

Column Name	Data Type	Description
JOB_USER	VARCHAR(10)	The user profile name of the user that started the job.
JOB_NUMBER	VARCHAR(6)	The job number of the job that added the entry.
THREAD	BIGINT	Identifies the thread within the process that added the journal entry.
PROGRAM_NAME	VARCHAR(10)	The name of the program that added the entry.
PROGRAM_LIBRARY	VARCHAR(10)	The name of the library that contains the program that added the journal entry.
PROGRAM_LIBRARY_ASP_DEVICE	VARCHAR(10)	The name of the ASP device that contains the program.
PROGRAM_LIBRARY_ASP_NUMBER	INTEGER	The number for the auxiliary storage pool that contains the program that added the journal entry.
COMMIT_CYCLE	DECIMAL(21,0)	A number that identifies the commit cycle.
NESTED_COMMIT_LEVEL	BIGINT	Indicates the nesting level of the commit cycle that was open when a journal entry representing an object level change was deposited.
XID	VARCHAR(140)	The transaction identifier, as defined by the Open Group's XA specification, for commit cycles related to an XA transaction branch.
LUW	VARCHAR(39)	The logical unit of work identifies entries to be associated with a given unit of work.
REMOTE_PORT	INTEGER	The port number of the remote address associated with this journal entry.
REMOTE_ADDRESS	VARCHAR(46)	The remote address associated with the journal entry.
SYSTEM_NAME	VARCHAR(8)	The name of the system on which the entry is being retrieved.
SYSTEM_SEQUENCE_NUMBER	DECIMAL(21,0)	The system sequence number indicates the relative sequence of when this journal entry was deposited into the journal.
REFERENTIAL_CONSTRAINT	CHAR(1)	Whether this entry was recorded for actions that occurred on records that are part of a referential constraint.
TRIGGER	CHAR(1)	Whether this entry was created as result of a trigger program.
IGNORE_ON_APPLY	CHAR(1)	Whether this entry is ignored during an Apply Journaled Changes (APYJRNCHG) or Remove Journaled Changed (RMVJRNCHG) command.
MINIMIZED_ENTRY_DATA	CHAR(1)	Whether this entry has minimized entry specific data as a result of the journal having specified MINENTDTA for the object type of the entry.
MINIMIZED_ON_FIELD_BOUNDARY	CHAR(1)	Whether this entry has minimized entry specific data on field boundaries as a result of the journal having been specified with MINENTDTA(*FLDBDY).
INDICATOR_FLAG	CHAR(1)	An indicator for the operation.
RECEIVER_NAME	VARCHAR(10)	The name of the receiver holding the journal entry.
RECEIVER_LIBRARY	VARCHAR(10)	The name of the library containing the receiver holding the journal entry.
RECEIVER_ASP_DEVICE	VARCHAR(10)	The name of the ASP device containing the receiver holding the journal entry.
RECEIVER_ASP_NUMBER	INTEGER	The number for the auxiliary storage pool containing the receiver holding the journal entry.
ARM_NUMBER	INTEGER	The number of the disk arm that contains the journal entry.
OBJECT_ASP_DEVICE	VARCHAR(10)	ASP device name.
OBJECT_ASP_NUMBER	INTEGER	ASP number.
PARENT_FILE_ID	BINARY(16)	File ID for parent directory.
OBJECT_FILE_ID	BINARY(16)	File ID for object.
RELATIVE_DIRECTORY_FILE_ID	BINARY(16)	File ID of directory containing object in PATH_NAME.
OBJECT_FILE_NAME	VARGRAPHIC(2002) CCSID 1200	Object name.
PATH_NAME	DBCLOB(16M) CCSID 1200	Name of IFS path.
DLO_NAME	VARCHAR(12)	DLO name.
FOLDER_PATH	VARCHAR(63)	DLO folder path.

Table 80. DISPLAY_JOURNAL table function (continued)

Column Name	Data Type	Description
SYSLOG_EVENT	VARGRAPHIC(2048) CCSID 1200	The Common Event Format (CEF) syslog event for the journal entry preceded with a header of the requested type. If a header-type of RFC3164 is requested, the maximum length is 1024 characters. If a header-type of RFC5424 is requested, the maximum length is 2048 characters. The string will be truncated with no warning if it exceeds the maximum length. The audit journal entry types that generate syslog information and the key names returned for journal entries are listed in the Notes section. Contains the null value if there is no syslog event defined for the journal entry or if NO was specified for the GENERATE_SYSLOG parameter.
SYSLOG_FACILITY	INTEGER	The syslog facility assigned to the event. **4** security/authorization messages Contains the null value if there is no syslog event defined for the journal entry or if NO was specified for the GENERATE_SYSLOG parameter.
SYSLOG_SEVERITY	INTEGER	The syslog severity assigned to the event. **2** Critical condition **4** Warning condition **5** Notice: A normal but significant condition **6** Informational message The severity assigned to each journal entry is listed in the Notes section. Contains the null value if there is no syslog event defined for the journal entry or if NO was specified for the GENERATE_SYSLOG parameter.
SYSLOG_PRIORITY	INTEGER	The syslog priority number assigned to the event. Contains the null value if there is no syslog event defined for the journal entry or if NO was specified for the GENERATE_SYSLOG parameter.

Note

Row and column access control: This table function recognizes whether ROW ACCESS CONTROL or COLUMN ACCESS CONTROL exists and is activated for the target table. If any row or column access control is active for the table, the rule text logic defined for the row permissions and/or column masks is applied before returning the value in ENTRY_DATA. When the rule text for a row permission determines that the user invoking the function should not see the row, the ENTRY_DATA column contains the text `NOT AUTHORIZED`. If the user is allowed to see the row and a column mask exists, the rule text for the column mask determines the value returned for ENTRY_DATA.

Syslog information: Syslog information is returned for a subset of audit journal entries. Syslog information is also available for history log messages. See HISTORY_LOG_INFO table function for more details.

The following audit journal entries can generate syslog information:

AD
: Auditing changes

AF
: Authority failure

AX
: Row and column access control

CA
: Authority changes

CD
: Command string audit

CO
: Create object

CP
: User profile changed, created, or restored

DO
: Delete object

DS
: DST security password reset

GR
: Generic record

GS
: Socket description was given to another job

LD
: Link, unlink, or look up directory entry

OM
: Object move or rename

OR
: Object restore

OW
: Object ownership changed

PA
: Program changed to adopt authority

PG
: Change of an object's primary group

PW
: Invalid password

RA
: Authority change during restore

RJ
: Restoring job description with user profile specified

RO
: Change of object owner during restore

RP
: Restoring adopted authority program

RU
: Restoring user profile authority

RZ
: Changing a primary group during restore

SE
: Subsystem routing entry changed

SO
: Server security user information actions

ST
: Use of service tools

SV
: System value changed

ZC
: Object accessed (change)

ZR
: Object accessed (read)

The audit journal entries are assigned a SYSLOG_SEVERITY value in the following way:

- Severity 2 Critical condition
  - SV - System value when QAUDCTL is changed to *NONE
- Severity 4 Warning condition
  - AF - Authority failure
  - GR - Generic record, when function usage was checked and failed for a function name with a prefix of `QIBM_DB_`
- Severity 5 Notice: A normal but significant condition
  - AD - Auditing changes
  - AX - Row and column access control
  - CA - Authority changes
  - CP - User profile changed, created, or restored
  - DS - DST security password reset
  - OM - Object move or rename
  - OW - Object ownership changed
  - PG - Change of an object's primary group
  - PW - Invalid password
  - RA - Authority change during restore
  - RO - Change of object owner during restore
  - RU - Restoring user profile authority
  - RZ - Change a primary group during restore
  - SO - Server security user information actions
- Severity 6 Informational message
  - CD - Command string audit
  - CO - Create object
  - DO - Delete object
  - GR - Generic record, except for the Severity 4 case where function usage was checked and failed
  - GS - Socket description was given to another job
  - LD - Link, unlink, or look up directory entry
  - OR - Object restore
  - PA - Program changed to adopt authority
  - RJ - Restoring job description with user profile specified
  - RP - Restoring adopted authority program
  - SE - Subsystem routing entry changed
  - ST - Use of service tools
  - SV - System value changed, except for QAUDCTL severity 2 case
  - ZC - Object accessed (change)
  - ZR - Object accessed (read)

The Common Event Format key names that are generated within the SYSLOG_EVENT column are:

Table 81. Common Event Format key names

Common Event Format key name	Description
deviceExternalId	Device name (extracted from ENTRY_DATA column)
dloName	Document Library Object name (DLO_NAME column)
dloPath	Document Library Object folder path (FOLDER_PATH column)
dproc	Destination job (process) name (extracted from ENTRY_DATA column)
duser	Destination user name (extracted from ENTRY_DATA column)
filePath	IFS stream file path (PATH_NAME column)
fileType	Object type (OBJECT_TYPE column)
fname	IFS stream file name (OBJECT_FILE_NAME column)
msg	Additional information from the audit record not included in other keys (extracted from ENTRY_DATA column)
objName	Object name (OBJECT column)
oldDloName	Document Library Object name (before rename) (extracted from ENTRY_DATA column)
oldDloPath	Document Library Object folder path (before rename) (extracted from ENTRY_DATA column)
oldFileName	IFS stream file name (before rename) (extracted from ENTRY_DATA column)
oldFilePath	IFS stream file path (before rename) (extracted from ENTRY_DATA column)
oldObjName	Object name (before rename) (extracted from ENTRY_DATA column)
reason	Text description of the audit journal entry
shost	Source system (host) name (SYSTEM_NAME column)
sproc	Source job (process) name (JOB_NAME, JOB_USER, JOB_NUMBER columns)
spt	Source port number (REMOTE_PORT column)
src	Source IP address (REMOTE_ADDRESS column)
suser	Source user name (CURRENT_USER column)

Examples

- Select all entries from the *CURRENT receiver of journal TESTLIB/QSQJRN.

```
SELECT * FROM TABLE (
                 DISPLAY_JOURNAL( 'TESTLIB', 'QSQJRN')) AS JT;
```

- Find all changes made by SUPERUSER against the PRODDATA/SALES table. The first two arguments are passed without names since they correspond with the first two parameters for the function. The other four arguments are passed using the parameter name syntax to avoid specifying a value for the parameters that are not needed.

```
SELECT journal_code, journal_entry_type, object, object_type, X.*
FROM TABLE (
  QSYS2.Display_Journal(
            'PRODDATA', 'QSQJRN', -- Journal library and name
            OBJECT_LIBRARY=>'PRODDATA', OBJECT_NAME=>'SALES',
            OBJECT_OBJTYPE=>'*FILE', OBJECT_MEMBER=>'SALES'
    ) ) AS X
WHERE journal_entry_type in ('DL', 'PT', 'PX', 'UP') AND "CURRENT_USER" = 'SUPERUSER'
ORDER BY entry_timestamp DESC
```

- Select entries from the audit journal that return syslog information and format them with an RFC5424 header.

```
SELECT syslog_facility, syslog_severity, syslog_event
  FROM TABLE (QSYS2.DISPLAY_JOURNAL('QSYS', 'QAUDJRN',
                                    GENERATE_SYSLOG =>'RFC5424'
             ) ) AS X
  WHERE syslog_event IS NOT NULL;
```

JOURNAL_INFO view

The JOURNAL_INFO view contains information about journals, including remote journals.

The values returned for the columns in the view are closely related to the values returned by Retrieve Journal Information API. Refer to the API for more detailed information.

The following table describes the columns in the view. The schema is QSYS2.

Table 82. JOURNAL_INFO view

Column Name	System Column Name	Data Type	Description
JOURNAL_NAME	JRNNAME	VARCHAR(10)	The name of the journal.
JOURNAL_LIBRARY	SYS_DNAME	VARCHAR(10)	The name of the library that contains the journal.
ASP_NUMBER	ASPNUMBER	INTEGER	The number of the auxiliary storage pool to which storage for the journal is allocated.
JOURNAL_ASPGRP	JRNASPGRP	VARCHAR(10)	The name of the auxiliary storage pool (ASP) in which the journal resides. A value of *SYSBAS indicates the system ASP and all basic user ASPs.
ATTACHED_JOURNAL_RECEIVER_NAME	ATTRCVNAME	VARCHAR(10) Nullable	The name of the journal receiver that is currently attached to this journal. Contains the null value when there is no attached receiver.
ATTACHED_JOURNAL_RECEIVER_LIBRARY	ATTRCVLIB	VARCHAR(10) Nullable	The name of the library that contains the attached journal receiver. Contains the null value when there is no attached receiver.
MESSAGE_QUEUE	MSGQNAME	VARCHAR(10)	The name of the message queue that is associated with this journal.
MESSAGE_QUEUE_LIBRARY	MSGQLIB	VARCHAR(10)	The name of the library that contains the message queue.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
DELETE_RECEIVER_OPTION	DLTRCVOPT	VARCHAR(3)	Indicates whether the system deletes detached journal receivers that are associated with this journal when they are no longer needed for IPL or IASP vary on recovery. **YES** Detached journal receivers that are associated with this journal are deleted when they are no longer needed for IPL or IASP vary on recovery. **NO** Detached journal receivers that are associated with this journal are not deleted when they are no longer needed for IPL or IASP vary on recovery.
DELETE_RECEIVER_DELAY	DLTRCVDLY	INTEGER Nullable	The delay time (in minutes) between attempts to delete journal receivers associated with this journal. Contains the null value when DELETE_RECEIVER_OPTION is NO.
JOURNAL_TYPE	TYPE	VARCHAR(10)	The scope of the journal and some of its characteristics. ***LOCAL** This is a local journal. ***REMOTE** This is a remote journal.
JOURNAL_STATE	STATE	VARCHAR(10)	An indication as to whether journal entries are currently being sent to a journal. For a remote journal, this is whether the journal is actively receiving journal entries from the source system journal. ***INACTIVE** If this is a remote journal, this means journal entries cannot be received from a source journal. ***ACTIVE** If this is a local journal, this means journal entries can be deposited to this journal. If this is a remote journal, this means journal entries can be received from a source journal. ***FAILED** If this is a remote journal, this means journal entries cannot be received from a source journal due to a remote journal function failure. Does not apply to local journals. ***INACTPEND** If this is a remote journal, this means a request is being processed to set the journal state to *INACTIVE. Does not apply to local journals. ***STANDBY** If this is a local journal, this means that most journal entries are not deposited into the journal and there will be no errors indicating that the entry was not deposited. Does not apply to remote journals. ***CTLINACT** The remote journal is in the process of a controlled inactivate. ***PENDING** The remote journal is transitioning from an *INACTIVE state to an *ACTIVE state.
NUMBER_JOURNAL_RECEIVERS	NUMJRNRCV	INTEGER	The total number of journal receivers that are associated with the journal.
TOTAL_SIZE_JOURNAL_RECEIVERS	SIZJRNRCV	BIGINT	The total size of the journal receivers (in kilobytes) associated with the journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
NUMBER_REMOTE_JOURNALS	RMTJRNS	INTEGER	The total number of remote journals that are directly downstream of this journal.
REDIRECTED_RECEIVER_LIBRARY	RDRRCVLIB	VARCHAR(10) Nullable	For a local or *TYPE1 remote journal, the redirected receiver library name that is currently in place on this journal's local journal for any downstream journal receivers associated with *TYPE1 remote journals. Contains *NONE if no *TYPE1 remote journals have been added or if no receiver library redirection was specified when *TYPE1 remote journals were added. Contains the redirected receiver library name that is currently in place on this remote journal if the specified journal is a *TYPE2 remote journal. Contains the null value if no *TYPE1 remote journals have been added.
MAXIMUM_REMOTE_JOURNALS _ENTRIES_BEHIND	MAXRMTENTB	INTEGER Nullable	The maximum number of entries that are waiting to be sent to the target system for any remote journal. Contains the null value if NUMBER_REMOTE_JOURNALS is 0 or if no attached remote journals are active with async delivery mode.
MAXIMUM_REMOTE_JOURNALS _TIME_BEHIND	MAXRMTSECB	BIGINT Nullable	The maximum value (in hundredths of seconds) that the source journal is behind in sending journal entries to the target system for any remote journal. Contains the null value if NUMBER_REMOTE_JOURNALS is 0 or if no attached remote journals are active with async delivery mode.
MAXIMUM_REMOTE_JOURNALS _RETRANSMISSIONS	MAXRMTRETR	BIGINT Nullable	The maximum value for any remote journal of the total number of times the local system retransmitted a segment because an acknowledgement was not received. Contains the null value if NUMBER_REMOTE_JOURNALS is 0 or if no attached remote journals are active using TCP/IP.
JOURNAL_TEXT	TEXT	VARCHAR(50) Nullable	The text description of the journal. Contains the null value if the journal has no text.
MANAGE_RECEIVER_OPTION	MNGRCVOPT	VARCHAR(10) Nullable	Indicates whether the system or user manages the changing of journal receivers. ***SYSTEM** The system manages the changing of journal receivers. ***USER** The user manages the changing of journal receivers. Contains the null value for a remote journal.
MANAGE_RECEIVER_DELAY	MNGRCVDLY	INTEGER Nullable	The delay time (in minutes) between attempts to attach new journal receivers to this journal. Contains the null value when MANAGE_RECEIVER_OPTION is *USER or the null value.
REMOVE_INTERNAL_ENTRIES	RMVINTENT	VARCHAR(3) Nullable	Handling of internal system entries. **YES** The size of the attached receivers is reduced by automatic removal of the internal system entries. **NO** The internal system entries are not removed. Contains the null value for a remote journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
REMOVE_FIXED_LENGTH_DETAIL	MINFIXLEN	VARCHAR(3) Nullable	Handling of fixed length details. **YES** The size of the journal entries that are deposited into the attached journal receivers is reduced by the automatic removal of all fixed length data such as job name, machine sequence number, and so on. **NO** Fixed length data is not removed. Contains the null value for a remote journal.
RECEIVER_MAXIMUM_SIZE	MAXOPT	VARCHAR(10) Nullable	The receiver size option that applies to this journal receiver. ***MAXOPT1** The journal receivers attached to the journal can have a maximum receiver size of approximately one terabyte (1,099,511,627,776 bytes) and a maximum sequence number of 9,999,999,999. Additionally, the maximum size of the journal entry that can be deposited is 15,761,440 bytes. ***MAXOPT2** The journal receivers attached to the journal can have a maximum receiver size of approximately one terabyte (1,099,511,627,776 bytes) and a maximum sequence number of 9,999,999,999. Additionally, the maximum size of the journal entry which can be deposited is 4,000,000,000 bytes. ***MAXOPT3** The journal receivers attached to the journal can have a maximum receiver size of approximately one terabyte (1,099,511,627,776 bytes) and a maximum sequence number of 18,446,744,073,709,551,600. Additionally, the maximum size of the journal entry which can be deposited is 4,000,000,000 bytes. ***NONE** The journal receivers attached to the journal can have a maximum journal receiver size of approximately 1.9 gigabytes and a maximum sequence number of 2,147,483,136. Contains the null value for a remote journal.
MINIMIZE_ESD_FOR_DATA_AREAS	MINDTAARA	VARCHAR(3) Nullable	Indicates whether journal entries for data areas may have minimized entry specific data. **YES** Journal entries for data areas have minimized entry specific data. **NO** Journal entries for data areas do not have minimized entry specific data. Contains the null value for a remote journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MINIMIZE_ESD_FOR_FILES	MINFILE	VARCHAR(19) Nullable	Indicates whether journal entries for files may have minimized entry specific data. **NO** Journal entries for files will have complete entry specific data. **MINIMIZED** Journal entries for files may have minimized entry specific data. The minimizing does not occur on field boundaries. Therefore, the entry specific data may not be viewable and may not be used for auditing purposes. **MINIMIZED FOR AUDIT** Journal entries for files may have minimized entry specific data. The minimizing occurs on field boundaries. Therefore, the entry specific data will be viewable and may be used for auditing purposes. Contains the null value for a remote journal.
JOURNAL_CACHE	JRNCACHE	VARCHAR(3) Nullable	Specifies whether journal entries are cached before being written out to disk. **YES** Journal entries are cached before being written out to disk. **NO** Journal entries are not cached before being written out to disk. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _JOB_NAME	FLDJOB	VARCHAR(3) Nullable	Indicates whether the job name will be stored when journal entries are deposited. **YES** The job name will be stored when journal entries are deposited. **NO** The job name will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _USER_NAME	FLDUSR	VARCHAR(3) Nullable	Indicates whether the user name will be stored when journal entries are deposited. **YES** The user name will be stored when journal entries are deposited. **NO** The user name will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _PROGRAM_NAME	FLDPGM	VARCHAR(3) Nullable	Indicates whether the program name will be stored when journal entries are deposited. **YES** The program name will be stored when journal entries are deposited. **NO** The program name will not be stored when journal entries are deposited. Contains the null value for a remote journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
FIXED_LENGTH_DATA_INCLUDES _PROGRAM_LIBRARY	FLDPGMLIB	VARCHAR(3) Nullable	Indicates whether the program library name will be stored when journal entries are deposited. **YES** The program library name will be stored when journal entries are deposited. **NO** The program library name will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _SYSTEM_SEQUENCE_NUMBER	FLDSYSSEQ	VARCHAR(3) Nullable	Indicates whether the system sequence number will be stored when journal entries are deposited. **YES** The system sequence number will be stored when journal entries are deposited. **NO** The system sequence number will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _REMOTE_ADDRESS	FLDRMTADR	VARCHAR(3) Nullable	Indicates whether the remote address will be stored when journal entries are deposited. **YES** The remote address will be stored when journal entries are deposited. **NO** The remote address will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _THREAD_ID	FLDTHD	VARCHAR(3) Nullable	Indicates whether the thread identifier will be stored when journal entries are deposited. **YES** The thread identifier will be stored when journal entries are deposited. **NO** The thread identifier will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _LOGICAL_UNIT_OF_WORK_ID	FLDLUW	VARCHAR(3) Nullable	Indicates whether the logical unit of work identifier will be stored when journal entries are deposited. **YES** The logical unit of work identifier will be stored when journal entries are deposited. **NO** The logical unit of work identifier will not be stored when journal entries are deposited. Contains the null value for a remote journal.
FIXED_LENGTH_DATA_INCLUDES _TRANSACTION_ID	FLDXID	VARCHAR(3) Nullable	Indicates whether the transaction identifier will be stored when journal entries are deposited. **YES** The transaction identifier will be stored when journal entries are deposited. **NO** The transaction identifier will not be stored when journal entries are deposited. Contains the null value for a remote journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
JOURNALED_OBJECT_LIMIT	JRNOBJLMT	VARCHAR(10) Nullable	The number of objects that can be journaled to the journal. ***MAX250K** The maximum number of objects that can be journaled to the journal is 250,000. ***MAX10M** The maximum number of objects that can be journaled to the journal is 10,000,000. Contains the null value for a remote journal.
JOURNALED_OBJECTS	JRNALL	INTEGER Nullable	Total of all objects journaled to the journal. This count includes explicitly journaled objects such as files, file members, access paths, data areas, data queues, libraries, and integrated file system objects. This count also includes implicitly journaled objects such as journal receivers, commitment definitions, and objects journaled for system recovery purposes. Contains the null value for a remote journal.
JOURNALED_FILES	JRNFILE	INTEGER Nullable	The total number of files that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNALED_MEMBERS	JRNMBR	INTEGER Nullable	The total number of file members that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNALED_DATA_AREAS	JRNDTAARA	INTEGER Nullable	The total number of data areas that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNALED_DATA_QUEUES	JRNDTAQ	INTEGER Nullable	The total number of data queues that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNALED_IFS_OBJECTS	JRNIFS	INTEGER Nullable	The total number of integrated file system objects of type *DIR, *STMF, and *SYMLNK that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNALED_ACCESS_PATHS	JRNAP	INTEGER Nullable	The total number of access paths that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNALED_COMMITMENT_DEFINITIONS	JRNCMTDFN	INTEGER Nullable	The total number of commitment definitions that are currently being implicitly journaled to this journal. Contains the null value for a remote journal.
JOURNALED_LIBRARIES	JRNLIB	INTEGER Nullable	The total number of libraries that are currently being journaled to this journal. Contains the null value for a remote journal.
JOURNAL_RECOVERY_COUNT	JRNRCYCNT	INTEGER Nullable	The approximate number of journaled changes that would need to be recovered during journal synchronization for this journal in the event of an abnormal IPL or vary on. Contains the null value for a local journal with the value *SYSDFT or for a remote journal,
REMOTE_JOURNAL_TYPE	RMTJRNTYPE	VARCHAR(10) Nullable	The type of remote journal. Values are *TYPE1 and *TYPE2. Contains the null value for a local journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
JOURNAL_DELIVERY_MODE	DELIVMODE	VARCHAR(10) Nullable	The journal delivery mode that is being used to replicate journal entries to this journal. ***ASYNC** Journal entries are being delivered or replicated asynchronously. ***SYNC** Journal entries are being delivered or replicated synchronously. ***ASYNCPEND** Journal entries are to be delivered or replicated asynchronously, but the journal is currently in catch-up mode. ***SYNCPEND** Journal entries are to be delivered or replicated synchronously, but the journal is currently in catch-up mode. Contains the null value for a local journal or a remote journal whose JOURNAL_STATE field is not *ACTIVE or *CTLINACT.
LOCAL_JOURNAL_NAME	LCLJRNNAME	VARCHAR(10) Nullable	The journal name of the local journal. The local journal is the journal that is the initiator of the original journal deposit that has been replicated downstream to this journal. Contains the null value for a local journal.
LOCAL_JOURNAL_LIBRARY	LCLJRNLIB	VARCHAR(10) Nullable	The library name of the local journal. Contains the null value for a local journal.
LOCAL_JOURNAL_SYSTEM	LCLJRNSYS	VARCHAR(8) Nullable	The name of the system for the local journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
LOCAL_JOURNAL_ASPGRP	LCLASPGRP	VARCHAR(10) Nullable	The name of the independent auxiliary storage pool (ASP) group of the local journal. *SYSBAS is used to indicate the system ASP and all basic user ASPs. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
SOURCE_JOURNAL_NAME	SRCJRNNAME	VARCHAR(10) Nullable	The journal name of the source journal. The source journal is the journal that is directly upstream of this journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
SOURCE_JOURNAL_LIBRARY	SRCJRNLIB	VARCHAR(10) Nullable	The library name of the source journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
SOURCE_JOURNAL_SYSTEM	SRCJRNSYS	VARCHAR(8) Nullable	The name of the system for the source journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
SOURCE_JOURNAL_ASPGRP	SRCASPGRP	VARCHAR(10) Nullable	The name of the independent auxiliary storage pool (ASP) group of the source journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.

Table 82. JOURNAL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LOCAL_RECEIVER_SYSTEM	LCLRCVSYS	VARCHAR(8) Nullable	If this journal receiver is associated with a remote journal, the name of the system for the local journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
SOURCE_RECEIVER_SYSTEM	SRCRCVSYS	VARCHAR(8) Nullable	If this journal receiver is associated with a remote journal, the name of the system for the source journal. Contains *UNKNOWN if journal is a remote journal and does not have an attached receiver. Contains the null value for a local journal.
ACTIVATION_TIME	ACTDT	TIMESTAMP Nullable	If the journal is a remote journal and it is currently active, the date and time the journal was activated. Contains the null value for a local journal or a remote journal whose JOURNAL_STATE field is not *ACTIVE or *CTLINACT.
ESTIMATED_TIME_BEHIND	ESTBEHIND	BIGINT Nullable	If the journal is an active remote journal and the delivery mode is asynchronous, this is the estimated amount of time, in milliseconds, between when the journal entries are written to disk on the source system and when they are received on the target system. Contains the null value for a local journal or a remote journal whose JOURNAL_STATE field is not *ACTIVE or *CTLINACT.
MAXIMUM_TIME_BEHIND	MAXBEHIND	BIGINT Nullable	The maximum value of ESTIMATED_TIME_BEHIND since the journal was activated. Contains the null value for a local journal or a remote journal whose JOURNAL_STATE field is not *ACTIVE or *CTLINACT.
MAXIMUM_BEHIND_TIMESTAMP	MAXBHNDTIM	TIMESTAMP Nullable	The date and time that the ESTIMATED_TIME_BEHIND occurred. Contains the null value for a local journal or a remote journal whose JOURNAL_STATE field is not *ACTIVE or *CTLINACT.
JOURNAL_ENTRY_FILTERING	FILTER	VARCHAR(3) Nullable	Indicates whether or not journal entry filtering is active for this journal. **YES** Journal entry filtering is active for this journal. **NO** Journal entry filtering is not active for this journal. Contains the null value for a local journal or a remote journal whose JOURNAL_STATE field is not *ACTIVE or *CTLINACT.

Examples

- List all journals that are falling behind sending entries to one or more remote journals:

```
SELECT JOURNAL_NAME, JOURNAL_LIBRARY FROM QSYS2.JOURNAL_INFO,
     MAXIMUM_REMOTE_JOURNALS_ENTRIES_BEHIND,
     MAXIMUM_REMOTE_JOURNALS_TIME_BEHIND, MAXIMUM_REMOTE_JOURNALS_RETRANSMISSIONS
        WHERE MAXIMUM_REMOTE_JOURNALS_ENTRIES_BEHIND > 0
        ORDER BY MAXIMUM_REMOTE_JOURNALS_ENTRIES_BEHIND DESC
```

- Find any remote journals that are not currently active:

```
SELECT * FROM QSYS2.JOURNAL_INFO
         WHERE JOURNAL_TYPE = '*REMOTE'
```

```
            AND JOURNAL_STATE <> '*ACTIVE'
        ORDER BY JOURNAL_LIBRARY, JOURNAL_NAME,
```

- For security auditing reasons, find any journals that are not recording remote address info:

```
SELECT * FROM QSYS2.JOURNAL_INFO
         WHERE REMOVE_FIXED_LENGTH_DETAIL = 'YES'
             OR FIXED_LENGTH_DATA_INCLUDES_REMOTE_ADDRESS = 'NO'
```

Librarian Services

These services provide object and library list information.

LIBRARY_LIST_INFO view

The LIBRARY_LIST_INFO view contains information about the current job's library list.

The following table describes the columns in the view. The schema is QSYS2.

Table 83. LIBRARY_LIST_INFO view

Column Name	System Column Name	Data Type	Description
ORDINAL_POSITION	COLNO	INTEGER	Position of this entry in the library list.
SCHEMA_NAME	NAME	VARCHAR(128) Nullable	Name of the schema.
SYSTEM_SCHEMA_NAME	SYS_NAME	VARCHAR(10)	System name of the schema.
TYPE	TYPE	VARCHAR(15)	The portion of the library list containing the selected library. Possible values are: **USER** The library is in the user portion of the library list. **SYSTEM** The library is in the system portion of the library list. **PRODUCT** The library is a product library in the library list. **CURRENT** The library is the current library entry in the library list.
IASP_NUMBER	IASP	SMALLINT Nullable	The number of the auxiliary storage pool where storage is allocated for the library.
TEXT_DESCRIPTION	TEXT	VARGRAPHIC(50) CCSID 1200 Nullable	The text description of the library. Contains the null value is there is no text description.

Example

- See the current library list for your job

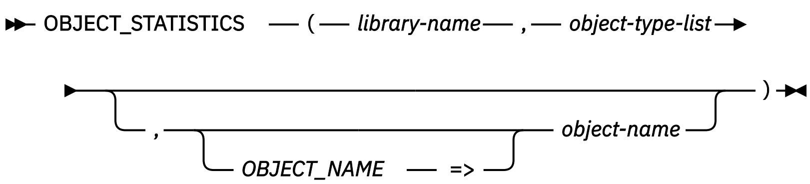
```
SELECT * FROM QSYS2.LIBRARY_LIST_INFO
```

OBJECT_STATISTICS table function

The OBJECT_STATISTICS table function returns information about objects in a library.

Authorization:

- If the user has *EXECUTE authority to the library, and both *OBJOPR and *READ authority to an object, full details are returned.
- Otherwise, partial information is returned along with an SQL warning of '01548'.



The schema is QSYS2.

library-name
A character or graphic string expression that identifies the name of a library. If the library's name is a delimited name, the delimited form of the name must be specified. It can be either a long or short library name.

The following special values are allowed for *library-name*.

***ALL**
All libraries.

***ALLUSR**
All user libraries in *SYSBAS and the current thread's ASP group.

***ALLUSRAVL**
All user libraries in all available ASPs.

***CURLIB**
The job's current library.

***LIBL**
The library list.

***USRLIBL**
The job's current library and the user portion of the library list.

The following special value is allowed for *library-name* when *object-type-list* is '*LIB' or 'LIB'.

***ALLSIMPLE**
The fastest approach to retrieving all user and system library names in *SYSBAS and the current thread's ASP group. Values are returned for the following columns: OBJNAME, OBJLONGNAME, and OBJTYPE. All other columns return NULL.

object-type-list
A character or graphic string expression containing one or more system object types separated by either a blank or a comma. The object types can include or exclude the leading * character. The special value of '*ALL' or 'ALL' can be used to return all objects in the library *library-name*.

object-name
A character or graphic string expression that identifies the name of an object or a library. If the object's name is a delimited name, the delimited form of the name must be specified. It can be either a long or short object name. The name must be the valid system name for the object unless the object is a file or a library; for files and libraries the SQL name can be specified.

If this parameter is specified, only objects with this name in *library-name* corresponding to the object types in *object-type-list* are returned.

If this parameter is not specified, all objects in *library-name* corresponding to the object types in *object-type-list* are returned.

The following special value is allowed for *object-name*.

***ALLSIMPLE**
The fastest approach to retrieving the system names for objects in a library. All objects in *library-name* corresponding to the object types in *object-type-list* are returned. Values are returned for the following columns: OBJNAME, OBJTYPE, and OBJLONGSCHEMA. All other columns return NULL.

The result of the function is a table containing a row for each object with the format shown in the following table. All the columns are null capable.

Table 84. OBJECT_STATISTICS table function

Column Name	Data Type	Description
OBJNAME	VARCHAR(10)	System name of the object.
OBJTYPE	VARCHAR(8)	System type of the object.
OBJOWNER	VARCHAR(10)	The user profile that owns the object.
OBJDEFINER	VARCHAR(10)	The user profile that created the object.
OBJCREATED	TIMESTAMP	Timestamp of when the object was created.
OBJSIZE	DECIMAL(15,0)	Size of the object, in bytes.
OBJTEXT	VARCHAR(50)	The description of the object. Contains the null value if the object has no text.
OBJLONGNAME	VARCHAR(128)	The SQL name for the object. For an external procedure or an external function, the name will be returned when a single procedure or function exists for that *PGM or *SRVPGM object. Contains the null value if an SQL name could not be returned.
LAST_USED_TIMESTAMP	TIMESTAMP	The date the object was used last. The time portion of the timestamp will always be 0. Contains the null value if the object has never been used.
DAYS_USED_COUNT	INTEGER	The number of days an object has been used on the system.
LAST_RESET_TIMESTAMP	TIMESTAMP	The date when the days used count was last reset to zero. The time portion of the timestamp will always be 0. Contains the null value if the days used count has never been reset.
IASP_NUMBER	SMALLINT	The auxiliary storage pool (ASP) where storage is allocated for the object.
OBJATTRIBUTE	VARCHAR(10)	The attribute for this object's type, if any. Contains an empty string if no attribute.
OBJLONGSCHEMA	VARCHAR(128)	The SQL schema name for this object.
TEXT	VARGRAPHIC(50) CCSID 1200	The description of the object, in CCSID 1200, for *LIB objects. Contains the null value if OBJTYPE is not *LIB.
SQL_OBJECT_TYPE	VARCHAR(9)	The SQL type of the object. Values are: • ALIAS • FUNCTION • INDEX • PACKAGE • PROCEDURE • ROUTINE • SEQUENCE • TABLE • TRIGGER • TYPE • VARIABLE • VIEW • XSR Contains the null value if the object is not an SQL object.

Example

- Find all journals in library MJATST.

```
SELECT * FROM TABLE (QSYS2.OBJECT_STATISTICS('MJATST ','JRN') ) AS X
```

or

```
SELECT * FROM TABLE (QSYS2.OBJECT_STATISTICS('MJATST ','*JRN') ) AS X
```

- Find all journals and journal receivers in library MJATST.

```
SELECT * FROM TABLE (QSYS2.OBJECT_STATISTICS('MJATST ','JRN JRNRCV') ) AS X
```

or

```
SELECT * FROM TABLE (QSYS2.OBJECT_STATISTICS('MJATST ','*JRN *JRNRCV') ) AS X
```

- Find all programs and service programs in library MYLIB. Use *ALLSIMPLE to return the list quickly, omitting the detail information.

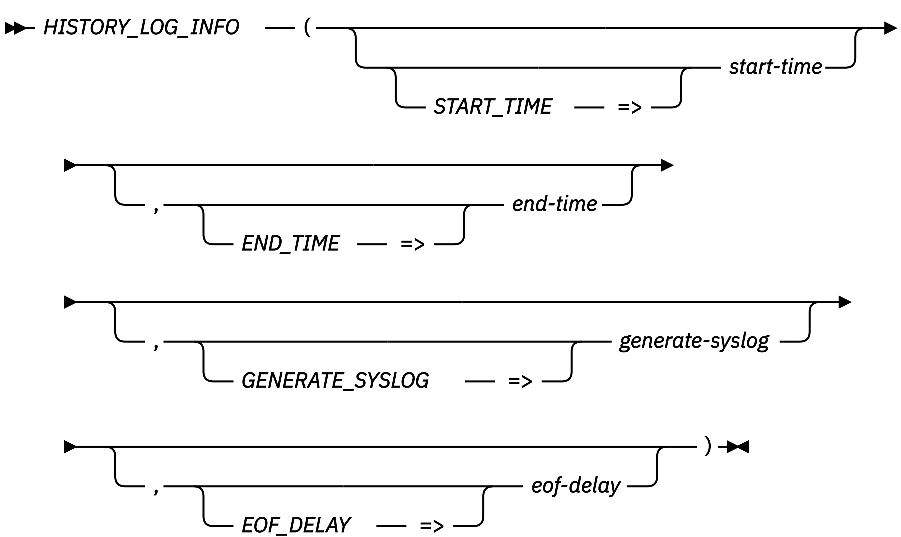
```
SELECT * FROM TABLE (QSYS2.OBJECT_STATISTICS('MYLIB','PGM SRVPGM','*ALLSIMPLE')) X
```

Message Handling Services

These views provide system message information.

HISTORY_LOG_INFO table function

The HISTORY_LOG_INFO table function returns one row for each message in the history log based on the timestamp range specified. It returns information similar to what is returned by the Display Log (DSPLOG) CL command and the Open List of History Log Messages (QMHOLHST) API.



The schema is QSYS2.

Authorization: No authorization needed.

start-time
A timestamp expression that indicates the starting timestamp to use when returning history log information.

If this parameter is omitted, the default of CURRENT DATE - 1 DAY is used.

end-time
A timestamp expression that indicates the ending timestamp to use when returning history log information.

If this parameter is omitted, the default of '9999-12-30-00.00.00.000000' is used.

generate-syslog
A character or graphic string expression that indicates whether to transform history log messages into syslog formatted detail. Values are:

NO

No syslog information will be returned. The SYSLOG_EVENT, SYSLOG_FACILITY, SYSLOG_SEVERITY, and SYSLOG_PRIORITY columns will contain the null value.

RFC3164

Values will be returned for the SYSLOG_EVENT, SYSLOG_FACILITY, SYSLOG_SEVERITY, and SYSLOG_PRIORITY columns for each history log message. The SYSLOG_EVENT column will contain a syslog header that matches the RFC3164 format as described by the Internet Engineering Task Force (IETF) Request For Comments (RFC) 3164.

RFC5424

Values will be returned for the SYSLOG_EVENT, SYSLOG_FACILITY, SYSLOG_SEVERITY, and SYSLOG_PRIORITY columns for each history log message. The SYSLOG_EVENT column will contain a syslog header that matches the RFC5424 format as described by the Internet Engineering Task Force (IETF) Request For Comments (RFC) 5424.

If *generate-syslog* is not specified or is the null value, NO is used.

eof-delay

An integer expression that specifies the number of seconds to sleep when all history log messages have been read. This delay allows the caller to establish a polling service that will continually return rows, sleeping for the specified interval whenever all messages have been processed.

A value of zero indicates no delay is used and a finite set of rows will be returned. A value greater than zero indicates that the table function will sleep, as needed, to wait for new history log messages and never end. If *eof-delay* is not specified or is the null value, zero is used.

If this parameter has a value greater than zero, the *generate-syslog* parameter must be RFC3164 or RFC5424, and the *end-time* parameter cannot be specified with a value other than its default value.

When using a non-zero *eof-delay* parameter, avoid using query clauses that depend on returning a finite number of rows. For example, using the FETCH FIRST n ROWS clause can cause the query to end when the requested number of rows has been satisfied. A query using the HISTORY_LOG_INFO function with a non-zero *eof-delay* parameter does not allow data to be copied (ALWCPYDTA(*NO)). This means that a query requiring a copy of data, such as one using an ORDER BY clause or UNION DISTINCT, will issue an error and not be allowed.

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 85. HISTORY_LOG_INFO table function

Column Name	Data Type	Description
ORDINAL_POSITION	INTEGER	A unique number for each row that indicates the time order of messages in the job log. The first (oldest) message returned from the history log will have a value of 1. Subsequent messages will have a value one greater than the previous message. Since these values are assigned when this catalog is queried, there will be no gaps in values.
MESSAGE_ID	VARCHAR(7)	The message ID for this message. Contains the null value if this is an impromptu message or MESSAGE_TYPE is REPLY.
MESSAGE_TYPE	VARCHAR(13)	Type of message. Values are: • COMPLETION • DIAGNOSTIC • ESCAPE • INFORMATIONAL • INQUIRY • NOTIFY • REPLY • REQUEST • SENDER

Table 85. HISTORY_LOG_INFO table function (continued)

Column Name	Data Type	Description
MESSAGE_SUBTYPE	VARCHAR(22)	Subtype of message. The values returned for REPLY messages: • FROM EXIT PROGRAM • FROM SYSTEM REPLY LIST • MESSAGE DEFAULT USED • NOT VALIDITY CHECKED • SYSTEM DEFAULT USED • VALIDITY CHECKED The value returned for some REQUEST messages: • WITH PROMPTING Contains the null value for other message types.
SEVERITY	SMALLINT	The severity assigned to the message.
MESSAGE_TIMESTAMP	TIMESTAMP	The timestamp when the message was sent.
FROM_USER	VARCHAR(10)	The current user of the job when the message was sent.
FROM_JOB	VARCHAR(28)	The qualified job name when the message was sent.
FROM_PROGRAM	VARCHAR(10)	The program that sent the message.
MESSAGE_LIBRARY	VARCHAR(10)	The name of the library containing the message file. Contains the null value if MESSAGE_ID is null.
MESSAGE_FILE	VARCHAR(10)	The message file containing the message. Contains the null value if MESSAGE_ID is null.
MESSAGE_TOKENS	VARCHAR(4096) FOR BIT DATA	The message token string. If the value is longer than 4096 characters, it will be truncated with no warning. Contains the null value if there are no message tokens.
MESSAGE_TEXT	VARGRAPHIC(1024) CCSID 1200	The first level text of the message including tokens, or the impromptu message text. Contains the null value if MESSAGE_ID is null or if the message file could not be accessed.
MESSAGE_SECOND_LEVEL_TEXT	VARGRAPHIC(4096) CCSID 1200	The second level text of the message including tokens. Contains the null value if MESSAGE_ID is null or if the message has no second level text or if the message file could not be accessed.
SYSLOG_EVENT	VARGRAPHIC(2048) CCSID 1200	The Common Event Format (CEF) syslog event for the message preceded by a header of the requested type. If a header-type of RFC3164 is requested, the maximum length is 1024 characters. If a header-type of RFC5424 is requested, the maximum length is 2048 characters. The string will be truncated with no warning if it exceeds the maximum length. The key names returned for history log information are listed in the Notes section. Contains the null value if NO was specified for the GENERATE_SYSLOG parameter.
SYSLOG_FACILITY	INTEGER	The syslog facility assigned to the event. **1** user-level messages **4** security/authorization messages The facility assigned is defined in the Notes section. Contains the null value if NO was specified for the GENERATE_SYSLOG parameter.

Table 85. HISTORY_LOG_INFO table function (continued)

Column Name	Data Type	Description
SYSLOG_SEVERITY	INTEGER	The syslog severity assigned to the event. **1** Alert: Action must be taken immediately **3** Error condition **4** Warning condition **5** Notice: A normal but significant condition **6** Informational message **7** Debug level message The severity assigned is listed in the Notes section. Contains the null value if NO was specified for the GENERATE_SYSLOG parameter.
SYSLOG_PRIORITY	INTEGER	The syslog priority number assigned to the event. Contains the null value if NO was specified for the GENERATE_SYSLOG parameter.

Notes

Syslog information: Syslog information is returned for all messages in the history log. Syslog information is also available for audit journal entries. See DISPLAY_JOURNAL table function for more details.

All history log messages return a SYSLOG_FACILITY value of 1 except as noted below. Messages are assigned a SYSLOG_SEVERITY value in the following way:

- Severity 1 Alert: Action must be taken immediately
  - MESSAGE_TYPE contains a value of INQUIRY, NOTIFY, or REPLY
- Severity 3 Error condition
  - MESSAGE_ID contains a value of CPF1164 with a job ending code value in the MESSAGE_TEXT column of 30 or higher
  - MESSAGE_TYPE contains a value of ESCAPE when the SEVERITY column contains a value of 50 or greater
- Severity 4 Warning condition
  - MESSAGE_ID contains a value of CPF1393. The SYSLOG_FACILITY column is set to 4.
  - MESSAGE_ID contains a value of CPF1164 with a job ending code value in the MESSAGE_TEXT column of 20
  - MESSAGE_TYPE contains a value of ESCAPE when the SEVERITY column contains a value of 30 or greater but less than 50
- Severity 5 Notice: A normal but significant condition
  - MESSAGE_ID contains a value of CPF1164 with a job ending code value in the MESSAGE_TEXT column of 10
  - MESSAGE_TYPE contains a value of INFORMATIONAL, COMPLETION, DIAGNOSTIC, or REQUEST when the SEVERITY column contains a value of 50 or greater
- Severity 6 Informational message
  - MESSAGE_ID contains a value of CPF1164 with a job ending code value in the MESSAGE_TEXT column of 0
  - MESSAGE_TYPE contains a value of ESCAPE when the SEVERITY column contains a value less than 30
  - MESSAGE_TYPE contains a value of SENDER

- MESSAGE_TYPE contains a value of INFORMATIONAL, COMPLETION, DIAGNOSTIC, or REQUEST when the SEVERITY column contains a value less than 50
- Severity 7 Debug level message
  - MESSAGE_ID contains a value of CPF9897 or CPF9898 (regardless of severity or message type)

The Common Event Format key names that are generated within the SYSLOG_EVENT column are:

Table 86. Common Event Format key names

Common Event Format key name	Description
msg	The message text (MESSAGE_TEXT column) from the history log message
reason	Text description of the history log message
sproc	The qualified job name (FROM_JOB column) from the history log message
suser	Current user name (FROM_USER column) from the history log message

Examples

- Return a list of history log messages for all of yesterday and today.

```
SELECT * FROM TABLE(QSYS2.HISTORY_LOG_INFO()) X
```

- Return a list of all history log messages for the last 24 hours.

```
SELECT * FROM TABLE(QSYS2.HISTORY_LOG_INFO(CURRENT TIMESTAMP - 1 DAY)) X
```

- Return history log information since the last IPL, assuming that the last IPL timestamp is in a global variable named LAST_IPL_TIME.

```
SELECT * FROM TABLE(QSYS2.HISTORY_LOG_INFO(LAST_IPL_TIME, CURRENT TIMESTAMP)) A
```

- Return syslog information formatted with an RFC3164 header for all history log messages from the start of today forward into the future. When all history log messages have been returned to the caller, the query will pause for 5 minutes (300 seconds) before checking again for messages.

```
SELECT syslog_facility, syslog_severity, syslog_event
  FROM TABLE (QSYS2.HISTORY_LOG_INFO(START_TIME => CURRENT DATE,
                                     GENERATE_SYSLOG =>'RFC3164',
                                     EOF_DELAY => 300
             ) ) AS X;
```

JOBLOG_INFO table function

The JOBLOG_INFO table function returns one row for each message in a job log.

►► JOBLOG_INFO — (— *job-name* —) ►◄

The schema is QSYS2.

job-name

A character or graphic string expression that identifies the qualified name of a job. The special value of '*' indicates the current job.

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 87. JOBLOG_INFO table function

Column Name	Data Type	Description
ORDINAL_POSITION	INTEGER	A unique number for each row that indicates the time order of messages in the job log. The first (oldest) message in the job log will have a value of 1. Subsequent messages will have a value one greater than the previous message. Since these values are assigned when this catalog is queried, there will be no gaps in values. There is no visibility of messages that have been deleted from the job log.
MESSAGE_ID	VARCHAR(7)	The message ID for this message. Contains the null value if this is an impromptu message or a REQUEST message.
MESSAGE_TYPE	VARCHAR(13)	Type of message. Values are: • COMMAND • COMPLETION • DIAGNOSTIC • ESCAPE • INFORMATIONAL • INQUIRY • NOTIFY • REPLY • REQUEST • SCOPE • SENDER
MESSAGE_SUBTYPE	VARCHAR(22)	Subtype of message. Values for NOTIFY or ESCAPE messages are: • EXCEPTION HANDLED • EXCEPTION NOT HANDLED Values for REPLY messages are: • FROM EXIT PROGRAM • FROM SYSTEM REPLY LIST • MESSAGE DEFAULT USED • NOT VALIDITY CHECKED • SYSTEM DEFAULT USED • VALIDITY CHECKED Contains the null value for other message types.
SEVERITY	SMALLINT	The severity assigned to the message.
MESSAGE_TIMESTAMP	TIMESTAMP	The timestamp for when the message was issued.
FROM_LIBRARY	VARCHAR(10)	The library containing the program or service program that sent the message.
FROM_PROGRAM	VARCHAR(256)	The program or service program name that sent the message.
FROM_MODULE	VARCHAR(10)	The module that sent the message.
FROM_PROCEDURE	VARCHAR(4096)	The procedure that sent the message.
FROM_INSTRUCTION	VARCHAR(10)	The instruction that sent the message.
TO_LIBRARY	VARCHAR(10)	The library containing the program or service program that received the message
TO_PROGRAM	VARCHAR(10)	The program or service program name that received the message.
TO_MODULE	VARCHAR(10)	The module that received the message.
TO_PROCEDURE	VARCHAR(4096)	The procedure that received the message.
TO_INSTRUCTION	VARCHAR(10)	The instruction that received the message.
FROM_USER	VARCHAR(10)	The userid of the job when the message was sent.
MESSAGE_FILE	VARCHAR(10)	The message file containing the message.
MESSAGE_LIBRARY	VARCHAR(10)	The name of the library containing the message file.
MESSAGE_TOKEN_LENGTH	SMALLINT	The length of the MESSAGE_TOKENS string.

Table 87. JOBLOG_INFO table function (continued)

Column Name	Data Type	Description
MESSAGE_TOKENS	VARCHAR(512) FOR BIT DATA	The message token string. If the value is longer than 512 characters, it will be truncated with no warning.
MESSAGE_TEXT	VARGRAPHIC(1024) CCSID 1200	The first level text of the message including tokens.
MESSAGE_SECOND_LEVEL_TEXT	VARGRAPHIC(4096) CCSID 1200	The second level text of the message including tokens.

Examples

- Return joblog information for job 347117/Quser/Qzdasoinit.

```
SELECT * FROM TABLE(QSYS2.JOBLOG_INFO('347117/Quser/Qzdasoinit')) A
```

- Extract the last command entered by the user.

```
SELECT MESSAGE_TEXT FROM TABLE(QSYS2.JOBLOG_INFO('817029/QUSER/QPADEV0004')) A
  WHERE A.MESSAGE_TYPE = 'REQUEST'
  ORDER BY ORDINAL_POSITION DESC
  FETCH FIRST 1 ROW ONLY
```

MESSAGE_QUEUE_INFO view

The MESSAGE_QUEUE_INFO view returns one row for each message in a message queue. It returns information similar to what is returned by the Display Messages (DSPMSG) CL command and the Receive Nonprogram Message (QMHRCVM) API.

This view does not change the contents of the message queue. The message is kept in the message queue without changing its new or old designation. The view does not utilize the wait time parameter as described in the QMHRCVM API. A wait time of 0 is used.

Authorization: The user must have *USE authority to the message queue and *EXECUTE authority to the message queue library.

The following table describes the columns in the view. The system name is MSGQ_INFO. The schema is QSYS2.

Table 88. MESSAGE_QUEUE_INFO view

Column Name	System Column Name	Data Type	Description
MESSAGE_QUEUE_LIBRARY	MSGQ_LIB	VARCHAR(10)	The name of the library containing the message queue.
MESSAGE_QUEUE_NAME	MSGQ_NAME	VARCHAR(10)	The name of the message queue containing the message.
MESSAGE_ID	MSGID	VARCHAR(7) Nullable	The message ID for this message. Contains the null value if this is an impromptu message or MESSAGE_TYPE is REPLY.
MESSAGE_TYPE	MSG_TYPE	VARCHAR(13)	Type of message. Values are: • COMPLETION • DIAGNOSTIC • ESCAPE • INFORMATIONAL • INQUIRY • NOTIFY • REPLY • REQUEST • SENDER

Table 88. MESSAGE_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MESSAGE_SUBTYPE	MSG_SUBTYP	VARCHAR(22) Nullable	Subtype of message. The values returned for REPLY messages: • FROM EXIT PROGRAM • FROM SYSTEM REPLY LIST • MESSAGE DEFAULT USED • NOT VALIDITY CHECKED • SYSTEM DEFAULT USED • VALIDITY CHECKED The value returned for some REQUEST messages: • WITH PROMPTING Contains the null value for other message types.
MESSAGE_TEXT	MSG_TEXT	VARGRAPHIC(1024) CCSID 1200 Nullable	The first level text of the message including tokens, or the impromptu message text. Contains the null value if MESSAGE_TYPE is REPLY or if the message file could not be accessed.
SEVERITY	SEVERITY	SMALLINT	The severity assigned to the message.
MESSAGE_TIMESTAMP	MSG_TIME	TIMESTAMP	The timestamp when the message was sent.
MESSAGE_KEY	MSG_KEY	BINARY(4)	The key assigned to the message. The key is assigned by the command or API that sends the message. For details, see Message Types and Message Keys in the Qmhrcvm API
ASSOCIATED_MESSAGE_KEY	ASSOC_KEY	BINARY(4) Nullable	For MESSAGE_TYPE of REPLY, contains the associated inquiry or notify message key. Contains the null value for other message types.
FROM_USER	FROM_USER	VARCHAR(10)	The current user of the thread when the message was sent.
FROM_JOB	FROM_JOB	VARCHAR(28)	The qualified job name of the job that sent the message.
FROM_PROGRAM	FROM_PGM	VARCHAR(10)	The program that sent the message.
MESSAGE_FILE_LIBRARY	MSGF_LIB	VARCHAR(10) Nullable	The name of the library containing the message file. Contains the null value if MESSAGE_ID is null.
MESSAGE_FILE_NAME	MSGF_NAME	VARCHAR(10) Nullable	The message file containing the message. Contains the null value if MESSAGE_ID is null.
MESSAGE_TOKENS	MSG_TOKENS	VARCHAR(4096) FOR BIT DATA Nullable	The message token string. If the value is longer than 4096 characters, it will be truncated with no warning. Contains the null value if there are no tokens.
MESSAGE_SECOND_LEVEL_TEXT	MSG_TEXT2	VARGRAPHIC(4096) CCSID 1200 Nullable	The second level text of the message including tokens. Contains the null value if MESSAGE_ID is null or if the message has no second level text or if the message file could not be accessed.

Example

- Examine all inquiry messages and their responses.

```
SELECT a.message_timestamp, a.message_text, a.from_user,
       b.message_timestamp, b.message_text, b.from_user
FROM qsys2.message_queue_info a INNER JOIN  qsys2.message_queue_info b
     ON a.message_key = b.associated_message_key
WHERE a.message_type = 'INQUIRY' AND
      b.message_type = 'REPLY'
ORDER BY b.message_timestamp DESC;
```

REPLY_LIST_INFO view

The REPLY_LIST_INFO view contains information about the current job's reply list entries.

The following table describes the columns in the view. The schema is QSYS2.

Table 89. REPLY_LIST_INFO view

Column Name	System Column Name	Data Type	Description
SEQUENCE_NUMBER	SEQNO	SMALLINT	The number that specifies the search order of the entries in the reply list.
MESSAGE_ID	MSGID	VARCHAR(7)	The identifier of the inquiry message for which automatic system action is to be taken. A value of ANY indicates that this reply list entry matches any message identifier. Unless comparison data is specified for this reply list entry, all reply list entries with a sequence number greater than this one are ignored.
MESSAGE_REPLY	REPLY	VARCHAR(32)	When an inquiry message is received with a matching message identifier, this value defines whether an automatic reply to the message is given. **DEFAULT** The default reply to the inquiry message is sent. **REQUIRED** The inquiry message requires an explicit reply. **character string** The character string to be sent as the reply to the inquiry message.
COMPARISON_DATA	COMPDATA	VARGRAPHIC(28) CCSID 1200 Nullable	The character string that is compared with the message data of the inquiry message. Contains the null value if there is no comparison data.
COMPARISON_DATA_OFFSET	OFFSET	SMALLINT Nullable	The position in the message data of the inquiry message at which the comparison with the COMARISON_DATA starts. Contains the null value if there is no comparison data.
DUMP_JOB	DUMPJOB	VARCHAR(3)	Specifies whether the job that sent the inquiry message is to be dumped. **NO** The job is not dumped. **YES** The job is dumped before control returns to the program that is sending the message.

Example

- See the reply list entries for your job

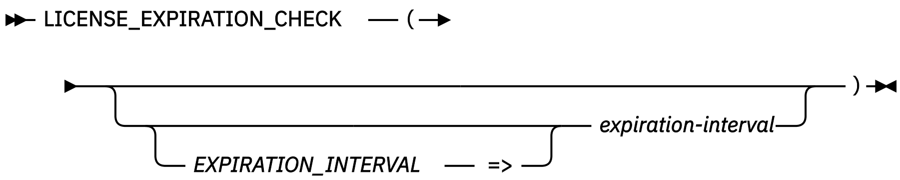
```
SELECT * FROM QSYS2.REPLY_LIST_INFO
```

Product Services

These services provide information about licensed products.

LICENSE_EXPIRATION_CHECK procedure

The LICENSE_EXPIRATION_CHECK procedure sends a message to the QSYSOPR message queue for every license that corresponds to an installed product that has already expired or is set to expire within the specified number of days.



The schema is SYSTOOLS.

expiration-interval
: An integer value that indicates the number of days to use as the threshold for checking license information. If not specified, 30 will be used.

Authorization: None required.

Example

- Send an informational message to the system operator message queue, QSYS/QSYSOPR, for every installed product with a license that will expire in the next 10 days.

```
CALL SYSTOOLS.LICENSE_EXPIRATION_CHECK(10);
```

LICENSE_INFO view

The LICENSE_INFO view contains information about all products or features that contain license information.

The values returned for the columns in the view are similar to the values returned by the Work with Licence Information (WRKLICINF) CL command or Retrieve License Information (QLZARTV) API. Refer to the API for more detailed information.

Authorization: None required.

The following table describes the columns in the view. The system name is LIC_INFO. The schema is QSYS2.

Table 90. LICENSE_INFO view

Column Name	System Column Name	Data Type	Description
PRODUCT_ID	LICPGM	VARCHAR(7)	The identifier of the product.
LICENSE_TERM	LIC_TERM	VARCHAR(6)	The license term indicates whether the authorized usage limit for a product exists until the next version, next release, or next modification level of the product. • Vx or vv for products licensed by version. • VxRy or vvrr for products licensed by release. • VxRyMz or vvrrmm for products licensed by modification.
RELEASE_LEVEL	RLS_LVL	VARCHAR(6)	The version, release, and modification level of the product in either VxRyMz or vvrrmm format.
FEATURE_ID	FEATURE	VARCHAR(4)	The feature number of the product.
INSTALLED	INSTALLED	VARCHAR(3)	Indicates whether this feature number of the product is installed. **NO** The feature is not installed. **YES** The feature is installed.
PROCESSOR_GROUP	PROC_GROUP	VARCHAR(3)	The processor group of this system.

Table 90. LICENSE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
PRODUCT_TEXT	LABEL	VARGRAPHIC(50) CCSID 1200 Nullable	The description of the product or feature. Contains null if there is no description text.
USAGE_LIMIT	USG_LIMIT	INTEGER Nullable	The usage limit of the product or feature that contains license information. Values are 0-999999 indicating the number of users allowed to access the product. Contains null if there is no usage limit.
USAGE_LIMIT_UPDATED	USG_UPDATE	TIMESTAMP(0) Nullable	The timestamp when the usage limit was last updated. Contains null if the usage limit has never been updated.
USAGE_TYPE	USAGE_TYPE	VARCHAR(11)	The usage type of the license. ***CONCURRENT** The usage type is concurrent, meaning the usage limit is for the number of uses held by unique jobs using the product or feature at the same time. ***REGISTERED** The usage type is registered, meaning the usage limit is for the number of uses held by license users registered to use the product or feature. ***PROCESSOR** The usage type is processor, meaning the usage limit is for the number of processors on system partitions where this product or feature is in use.
USAGE_COUNT	USG_COUNT	DECIMAL(8,2)	The current usage count for the product or feature. Valid values are 0 through 999999. If the product is using processor usage type, the usage count value is rounded up to the next whole number.
GLOBAL_COUNT	GLOB_COUNT	DECIMAL(8,2)	The number of jobs currently using this product or feature across all system partitions.
LICENSED_USER_COUNT	LIC_COUNT	INTEGER	The number of current license users.
THRESHOLD	THRESHOLD	DECIMAL(10,2) Nullable	The usage limit threshold for this product or feature. Contains null if there is no usage limit threshold.
PEAK_USAGE	PEAK_USAGE	DECIMAL(10,2)	For concurrent usage, the maximum number of uses held by license users of the product or feature at one time. For registered usage, the maximum number of uses that have been registered through license users for the product or feature. For processor usage, the maximum number of processors configured for this system partition while this product or feature was in use.
LAST_PEAK_USAGE	LAST_PEAK	TIMESTAMP(0) Nullable	The timestamp when the peak usage of the product or feature last occurred since the peak usage was reset to zero. Contains null if the product has not been used since the peak usage was reset to zero.
COMPLIANCE_TYPE	COMP_TYPE	VARCHAR(10)	The compliance type of the program determines the action taken when the usage limit of the product or feature is exceeded. ***OPRACTION** License requests are denied and failure messages are sent. ***WARNING** A warning message is sent. ***KEYED** Requests for licenses over the usage limit are allowed for the number of days in the product's grace period. Once the grace period ends, the license users holding uses over the usage limit will be released and no requests for uses over the limit will be granted until a new license key is received from the software provider. The expiration date is the date the license will expire. After the expiration date is reached, the default usage limit is in effect. When a request for a license is received after the usage limit has been reached, the system sends a warning message to the system operator message queue and to any additional message queues defined for the product.

Table 90. LICENSE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LOG_VIOLATION	LOG	VARCHAR(3)	Specifies whether or not requests exceeding the usage limit are logged in the QUSRSYS/QLZALOG journal. **NO** The requests for a license when the usage count is greater than or equal to the usage limit will not be logged. **YES** The requests for a license when the usage count is greater than or equal to the usage limit will be logged.
LICENSE_EXPIRATION	EXPIR_DATE	DATE Nullable	The date the license will expire. After the expiration date is reached, the usage limit is reset to the default usage limit. Contains null if the license has no expiration date.
GRACE_PERIOD	GRACE_PRD	INTEGER	The number of days a user has to obtain a new license key after a product or feature exceeds its usage limit.
GRACE_END	GRACE_END	DATE Nullable	The date the grace period expires. When a request for license uses exceeds the usage limit for a product or feature, the date the grace period expires is determined by adding the number of days in the grace period to the current date. Contains null if there is no grace period or the grace period has expired.
VENDOR_DATA	VENDOR	VARCHAR(8)	Information the vendor defined when the key was added using the Add License Key Information (ADDLICKEY) command.
MESSAGE_QUEUE1	MESSAGE_1	VARCHAR(10) Nullable	The name of the first message queue to which messages will be sent. Each of these message queues, in addition to the system operator message queue, will be sent a messages if one of the following occurs: • The usage count threshold is met. • A license request is made, and the usage count is equal to or greater than the usage limit. • The usage limit is changed. The messages sent include: • CPI9E10 - License usage limit changed for product &1. • CPI9E19 - Usage limit threshold exceeded. • CPI9E75 - Grace period will expire on &3. • CPI9E76 - Expiration date will be reached on &3. Contains null if there is no first message queue.
MESSAGE_QUEUE_LIBRARY1	LIBRARY_1	VARCHAR(10) Nullable	The library containing the first message queue. Contains null if there is no first message queue.
MESSAGE_QUEUE2	MESSAGE_2	VARCHAR(10) Nullable	The name of the second message queue to which messages will be sent. Contains null if there is no second message queue.
MESSAGE_QUEUE_LIBRARY2	LIBRARY_2	VARCHAR(10) Nullable	The library containing the second message queue. Contains null if there is no second message queue.
MESSAGE_QUEUE3	MESSAGE_3	VARCHAR(10) Nullable	The name of the third message queue to which messages will be sent. Contains null if there is no third message queue.
MESSAGE_QUEUE_LIBRARY3	LIBRARY_3	VARCHAR(10) Nullable	The library containing the third message queue. Contains null if there is no third message queue.
MESSAGE_QUEUE4	MESSAGE_4	VARCHAR(10) Nullable	The name of the fourth message queue to which messages will be sent. Contains null if there is no fourth message queue.
MESSAGE_QUEUE_LIBRARY4	LIBRARY_4	VARCHAR(10) Nullable	The library containing the fourth message queue. Contains null if there is no fourth message queue.

Table 90. LICENSE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MESSAGE_QUEUE5	MESSAGE_5	VARCHAR(10) Nullable	The name of the fifth message queue to which messages will be sent. Contains null if there is no fifth message queue.
MESSAGE_QUEUE_LIBRARY5	LIBRARY_5	VARCHAR(10) Nullable	The library containing the fifth message queue. Contains null if there is no fifth message queue.

Example

Return information about all licensed products and features that will expire within the next 2 weeks.

```
SELECT * FROM QSYS2.LICENSE_INFO
   WHERE LICENSE_EXPIRATION <= CURRENT DATE + 14 DAYS;
```

PTF Services

These views provide PTF information.

GROUP_PTF_CURRENCY view

The GROUP_PTF_CURRENCY is a view containing a query which implements a live comparison of the PTF Groups installed on the partition against the service levels listed on the IBM Preventive Service Planning website.

When queried, the view uses the XMLTable() and HTTPGETBLOB() table functions to consume a live XML feed from IBM Preventive Service Planning (PSP). If the partition cannot connect to the PSP website, the PTF_GROUP_CURRENCY column will contain PSP `INFORMATION NOT AVAILABLE`. When querying this view, the job CCSID cannot be 65535 or the query will fail.

The results of the query show which PTF Groups installed on the partition match the latest level made available by IBM and those which have a more recent version available.

The following table describes the columns in the view. The schema is SYSTOOLS.

Table 91. GROUP_PTF_CURRENCY view

Column name	System column name	Data type	Description
PTF_GROUP_CURRENCY	GRP_CRNCY	VARCHAR(46) Nullable	A description of the PTF group's status. Values returned are: **INSTALLED LEVEL IS CURRENT** Indicates that the PTF Group level installed matches the most current level available from IBM **CURRENT AT THE NEXT IPL** Indicates that the most current PTF Group level available from IBM is ready to be applied when the next IPL occurs. **UPDATE AVAILABLE** Indicates that a more recent PTF Group level is available from IBM **PSP INFORMATION NOT AVAILABLE** Indicates that the query is unable to connect to the external IBM PSP PTF Group level feed.
PTF_GROUP_ID	GRP_ID	CHAR(7) Nullable	The name of the PTF group.
PTF_GROUP_TITLE	GRP_TITLE	VARCHAR(1000) Nullable	The descriptive name of the PTF group.
PTF_GROUP_LEVEL_INSTALLED	GRP_LVL	INTEGER Nullable	The most recent level of this PTF Group installed on the partition.

Table 91. GROUP_PTF_CURRENCY view (continued)

Column name	System column name	Data type	Description
PTF_GROUP_LEVEL_AVAILABLE	GRP_IBMLVL	INTEGER Nullable	The PTF Group level which is available from IBM PSP.
PTF_GROUP_LAST_UPDATED_BY_IBM	GRP_LSTUPD	CHAR(10) Nullable	The date that IBM made the latest PTF Group level available. This is the character form of the date formatted as MM/DD/YYYY.
PTF_GROUP_RELEASE	GRP_RLS	VARCHAR(6) Nullable	The release level of the PTF Group. For example, 'R710' indicates IBM i 7.1 release level.
PTF_GROUP_STATUS_ON_SYSTEM	GRP_SYSSTS	VARCHAR(20) Nullable	This column will always contain the value 'INSTALLED'.

Notes

- The PSP website is:

  ```
  http://www.ibm.com/support/docview.wss?uid=nas4PSPbyNum&aid=1
  ```

  To determine the IP address for your geography, ping www.ibm.com.
- The PTF_GROUP_STATUS_ON_SYSTEM column is included in this view to demonstrate that it would be possible to create your own version of this query or view which includes information about PTF Groups that are loaded, but not installed.

Example

Compare the PTF Group service level detail, ordering the results from furthest behind to current.

```
SELECT * FROM SYSTOOLS.GROUP_PTF_CURRENCY
   ORDER BY PTF_GROUP_LEVEL_AVAILABLE - PTF_GROUP_LEVEL_INSTALLED DESC
```

Related reference

SYSTOOLS
SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

GROUP_PTF_DETAILS view

The GROUP_PTF_DETAILS is a view containing a query which implements a live comparison of the PTFs within PTF Groups installed on the partition against the service levels listed on the IBM Preventive Service Planning website.

When queried, the view uses the XMLTable() and HTTPGETBLOB() table functions to consume a live XML feed from IBM Preventive Service Planning (PSP). If the partition cannot connect to the PSP website, the query will fail with an SQL4302. When querying this view, the job CCSID cannot be 65535 or the query will fail.

The results of the query show which PTFs from all PTF Groups installed on the partition match the latest level made available by IBM and those which have a more recent version available.

The following table describes the columns in the view. The schema is SYSTOOLS.

Table 92. GROUP_PTF_DETAILS view

Column name	System column name	Data type	Description
PTF_GROUP_DESCRIPTION	GRPDESC	VARCHAR(100)	Description of the PTF group.
PTF_GROUP_NAME	GRPNAME	CHAR(7)	Name of the PTF group.

Table 92. GROUP_PTF_DETAILS view (continued)

Column name	System column name	Data type	Description
PTF_STATUS	PTF_STATUS	VARCHAR(11)	Status of the PTF. **PTF APPLIED** The PTF has been loaded and applied. **PTF LOADED** The PTF has been loaded but not applied. **PTF MISSING** The PTF does not exists on this partition.
PTF_PRODUCT_ID	LICPGM	VARCHAR(7)	The licensed program for this PTF.
PTF_IDENTIFIER	PTFID	VARCHAR(7)	The identifier of the PTF.
APAR_NAME	APAR_NAME	VARCHAR(7)	The APAR name associated with the PTF.
PTF_INCLUDED_IN_GROUP_DATE	PTF_DATE	VARCHAR(10)	The date that this PTF was first made available in a group PTF. Contains the character form of a date formatted as MM/DD/YY.
PTF_CUM_PACKAGE	PTF_CUMPKG	VARCHAR(8)	The identifier of the cumulative PTF package containing this PTF.
PTF_PRODUCT_DESCRIPTION	PRODDESC	VARCHAR(132) Nullable	Product description.
PTF_RELEASE_LEVEL	PTFRLS	VARCHAR(6) Nullable	The release level of the PTF.
PTF_PRODUCT_LOAD	PRODLOAD	VARCHAR(4) Nullable	The load ID of the product load for the PTF.
PTF_LOADED_STATUS	LOADSTAT	VARCHAR(19) Nullable	The current loaded status of the PTF. **NOT LOADED** The PTF has never been loaded. **LOADED** The PTF has been loaded. **APPLIED** The PTF has been applied. **PERMANENTLY APPLIED** The PTF has been applied permanently. **PERMANENTLY REMOVED** The PTF has been permanently removed. **DAMAGED** The PTF is damaged. An error occurred while applying the PTF. It needs to be reloaded and applied. **SUPERCEDED** The PTF is superseded. A PTF will have a status of superseded when one of the following situations occurs: • Another PTF with a more recent correction for the problem has been loaded on the system. The PTF ID that has been loaded can be found in the PTF_SUPERCEDED_BY_PTF column. • The PTF save file for another PTF with a more recent correction for the problem has been logged into *SERVICE on the system.
PTF_SAVE_FILE	SAVF	VARCHAR(3) Nullable	Indicates whether a save file exists for the PTF. **NO** The PTF has no save file. **YES** The PTF has a save file.

Table 92. GROUP_PTF_DETAILS view (continued)

Column name	System column name	Data type	Description
PTF_COVER_LETTER	COVER	VARCHAR(3) Nullable	Indicates whether a cover letter exists for the PTF. **NO** The PTF has no cover letter. **YES** The PTF has a cover letter.
PTF_ON_ORDER	ONORD	VARCHAR(3) Nullable	Indicates whether the PTF has been ordered. **NO** The PTF has not been ordered or has already been received. **YES** The PTF has been ordered.
PTF_IPL_ACTION	IPLACT	VARCHAR(19) Nullable	The action to be taken on this PTF during the next unattended IPL. **NONE** No action occurs at the next IPL. **TEMPORARILY APPLIED** The PTF is temporarily applied at the next IPL. **TEMPORARILY REMOVED** The PTF is temporarily removed at the next IPL. **PERMANENTLY APPLIED** The PTF is permanently applied at the next IPL. **PERMANENTLY REMOVED** The PTF is permanently removed at the next IPL.
PTF_ACTION_PENDING	ACTPEND	VARCHAR(3) Nullable	Indicates whether a required action has yet to be performed to make this PTF active. **NO** No required actions are pending for this PTF. **YES** A required action needs to occur for this PTF to be active. Check the Activation Instructions section of the cover letter to determine what the action is. If the PTF_ACTION_REQUIRED column is set to IPL and the activation instructions have been performed, then the PTF is active. However, this column will not be updated until the next IPL.
PTF_ACTION_REQUIRED	ACTREQ	VARCHAR(12) Nullable	Indicates whether an action is required to make this PTF active when it is applied. See the cover letter to determine what action needs to be taken. **NONE** No activation instructions are needed for this PTF. **EXIT PROGRAM** This PTF was shipped with activation instructions in the cover letter. This value is returned for all PTFs that have an exit program to update the status of the PTF after the activation instructions have been performed. **IPL** This PTF was shipped with activation instructions in the cover letter. No exit program exists to verify the activation instructions were performed.
PTF_IPL_REQUIRED	IPLREQ	VARCHAR(9) Nullable	Indicates whether an IPL is required to apply this PTF. **DELAYED** The PTF is delayed. The PTF must be applied during an IPL. **IMMEDIATE** The PTF is immediate. No IPL is needed to apply the PTF. **UNKNOWN** The type of the PTF is not known.

Table 92. GROUP_PTF_DETAILS view (continued)

Column name	System column name	Data type	Description
PTF_IS_RELEASED	RELEASED	VARCHAR(3) Nullable	Indicates whether the PTF save file is available for distribution to another system. This is set to YES only when the System Manager for IBM i licensed program is on the system and the product is supported. The PTF_SAVE_FILE column must have a value of YES before using the value in this column. **NO** The PTF save file cannot be distributed. **YES** The PTF save file is released and can be distributed to another system.
PTF_MINIMUM_LEVEL	MINLVL	VARCHAR(2) Nullable	The indicator of the lowest level of the product to which this PTF can be applied. The level can be AA to 99. Contains the null value if the product does not have a level.
PTF_MAXIMUM_LEVEL	MAXLVL	VARCHAR(2) Nullable	The indicator of the highest level of the product to which this PTF can be applied. The level can be AA to 99. Contains the null value if the product does not have a level.
PTF_STATUS_TIMESTAMP	STATTIME	TIMESTAMP Nullable	The date and time that the PTF status was last changed. Contains the null value when the status date and time is not available.
PTF_SUPERCEDED_BY_PTF	SUPERCEDE	VARCHAR(7) Nullable	The identifier of the PTF that has replaced this PTF. This field will be blank when the PTF is not superseded or when the superseding PTF has not been loaded on the system.
PTF_CREATION_TIMESTAMP	CRTTIME	TIMESTAMP Nullable	The date and time that the PTF was created. Contains the null value when the creation date and time cannot be determined.

Note

The PSP websites used by this service are found based upon the PTF groups that are currently installed on the partition. For each distinct PTF group, a unique PSP XML feed is accessed:

```
http://www.ibm.com/support/docview.wss?uid=nas4<PTF-Group-Name>&aid=1
```

For example, the JAVA PTF group details can be accessed using:

```
http://www.ibm.com/support/docview.wss?uid=nas4SF99572&aid=1
```

To determine the IP address for your geography, ping www.ibm.com.

Example

- Review the details for the PTFs which have not yet been applied for the PTF groups installed on this partition.

```
SELECT * FROM SYSTOOLS.GROUP_PTF_DETAILS
  WHERE PTF_STATUS <> 'PTF APPLIED'
  ORDER BY PTF_GROUP_NAME
```

Related reference

SYSTOOLS
SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

GROUP_PTF_INFO view

The GROUP_PTF_INFO view contains information about the group PTFs for the server.

The information returned is similar to the information available from Work with PTF Groups (WRKPTFGRP) CL command.

The following table describes the columns in the view. The schema is QSYS2.

Table 93. GROUP_PTF_INFO view

Column name	System column name	Data type	Description
COLLECTED_TIME	COLLE00001	TIMESTAMP	Date and time of when this row information was generated.
PTF_GROUP_NAME	PTF_G00001	VARCHAR(60) Nullable	Name of the PTF group.
PTF_GROUP_DESCRIPTION	PTF_G00002	VARCHAR(100) Nullable	Description of the PTF group.
PTF_GROUP_LEVEL	PTF_G00003	INTEGER Nullable	Level of the PTF group.
PTF_GROUP_TARGET_RELEASE	PTF_G00004	VARCHAR(6) Nullable	Release level for PTF group.
PTF_GROUP_STATUS	PTF_G00005	VARCHAR(20) Nullable	Status of the PTF group. **UNKNOWN** The PTF group status cannot be resolved because a related PTF group is either not found on the system or is in error. **NOT APPLICABLE** All PTFs in the PTF group and related PTF groups are for products that are not installed or supported on this system. **SUPPORTED ONLY** There are no PTFs in the PTF group or related PTF groups that are for installed products on this system. There is at least one PTF that is for a product, release, option, and load identifier that is supported on this system. **NOT INSTALLED** There is at least one PTF that is for an installed product on this system, and not all of the PTFs or their superseding PTFs are temporarily or permanently applied. **INSTALLED** All PTFs for products that are installed on this system are temporarily or permanently applied. If a PTF is superseded, a superseding PTF is either temporarily or permanently applied. **ERROR** The PTF group information is in error. Either delete the PTF group or replace the PTF group information that is currently on the system. **APPLY AT NEXT IPL** All PTFs for the installed products on the system are either set to be applied at the next IPL or are already temporarily or permanently applied. **RELATED GROUP** The PTF group does not have any PTFs for products installed or supported on the system. However, it is identified in another PTF group as a related PTF group. Deleting a PTF group in this status will cause the other PTF group to have a status of UNKNOWN. **ON ORDER** There is at least one PTF in the group that is on order and has not yet been installed on the system. It will be delivered on either physical or virtual media.

Example

Determine the level of the latest CUM PTF group installed on the system.

```
SELECT MAX(PTF_GROUP_LEVEL) AS CUM_LEVEL
  FROM QSYS2.GROUP_PTF_INFO
  WHERE PTF_GROUP_NAME IN ('SF99610','SF99710')
        AND PTF_GROUP_STATUS = 'INSTALLED'
```

PTF_INFO view

The PTF_INFO view contains information about PTFs for the server.

The information returned is similar to QpzListPTF API.

The following table describes the columns in the view. The schema is QSYS2.

Table 94. PTF_INFO view

Column name	System column name	Data type	Description
PTF_PRODUCT_ID	LICPGM	VARCHAR(7) Nullable	Product identifier.
PTF_PRODUCT_OPTION	PRODOPT	VARCHAR(6) Nullable	Product option.
PTF_PRODUCT_RELEASE_LEVEL	PRODRLS	VARCHAR(6) Nullable	Product release level.
PTF_PRODUCT_DESCRIPTION	PRODDESC	VARCHAR(132) Nullable	Product description.
PTF_IDENTIFIER	PTFID	VARCHAR(7) Nullable	The identifier of the PTF.
PTF_RELEASE_LEVEL	PTFRLS	VARCHAR(6) Nullable	The release level of the PTF.
PTF_PRODUCT_LOAD	PRODLOAD	VARCHAR(4) Nullable	The load ID of the product load for the PTF.
PTF_LOADED_STATUS	LOADSTAT	VARCHAR(19) Nullable	The current loaded status of the PTF. **NOT LOADED** The PTF has never been loaded. **LOADED** The PTF has been loaded. **APPLIED** The PTF has been applied. **PERMANENTLY APPLIED** The PTF has been applied permanently. **PERMANENTLY REMOVED** The PTF has been permanently removed. **DAMAGED** The PTF is damaged. An error occurred while applying the PTF. It needs to be reloaded and applied. **SUPERCEDED** The PTF is superseded. A PTF will have a status of superseded when one of the following situations occurs: • Another PTF with a more recent correction for the problem has been loaded on the system. The PTF ID that has been loaded can be found in the PTF_SUPERCEDED_BY_PTF column. • The PTF save file for another PTF with a more recent correction for the problem has been logged into *SERVICE on the system.

Table 94. PTF_INFO view (continued)

Column name	System column name	Data type	Description
PTF_SAVE_FILE	SAVF	VARCHAR(3) Nullable	Indicates whether a save file exists for the PTF. **NO** The PTF has no save file. **YES** The PTF has a save file.
PTF_COVER_LETTER	COVER	VARCHAR(3) Nullable	Indicates whether a cover letter exists for the PTF. **NO** The PTF has no cover letter. **YES** The PTF has a cover letter.
PTF_ON_ORDER	ONORD	VARCHAR(3) Nullable	Indicates whether the PTF has been ordered. **NO** The PTF has not been ordered or has already been received. **YES** The PTF has been ordered.
PTF_IPL_ACTION	IPLACT	VARCHAR(19) Nullable	The action to be taken on this PTF during the next unattended IPL. **NONE** No action occurs at the next IPL. **TEMPORARILY APPLIED** The PTF is temporarily applied at the next IPL. **TEMPORARILY REMOVED** The PTF is temporarily removed at the next IPL. **PERMANENTLY APPLIED** The PTF is permanently applied at the next IPL. **PERMANENTLY REMOVED** The PTF is permanently removed at the next IPL.
PTF_ACTION_PENDING	ACTPEND	VARCHAR(3) Nullable	Indicates whether a required action has yet to be performed to make this PTF active. **NO** No required actions are pending for this PTF. **YES** A required action needs to occur for this PTF to be active. Check the Activation Instructions section of the cover letter to determine what the action is. If the PTF_ACTION_REQUIRED column is set to IPL and the activation instructions have been performed, then the PTF is active. However, this column will not be updated until the next IPL.
PTF_ACTION_REQUIRED	ACTREQ	VARCHAR(12) Nullable	Indicates whether an action is required to make this PTF active when it is applied. See the cover letter to determine what action needs to be taken. **NONE** No activation instructions are needed for this PTF. **EXIT PROGRAM** This PTF was shipped with activation instructions in the cover letter. This value is returned for all PTFs that have an exit program to update the status of the PTF after the activation instructions have been performed. **IPL** This PTF was shipped with activation instructions in the cover letter. No exit program exists to verify the activation instructions were performed.

Table 94. PTF_INFO view (continued)

Column name	System column name	Data type	Description
PTF_IPL_REQUIRED	IPLREQ	VARCHAR(9) Nullable	Indicates whether an IPL is required to apply this PTF. **DELAYED** The PTF is delayed. The PTF must be applied during an IPL. **IMMEDIATE** The PTF is immediate. No IPL is needed to apply the PTF. **UNKNOWN** The type of the PTF is not known.
PTF_IS_RELEASED	RELEASED	VARCHAR(3) Nullable	Indicates whether the PTF save file is available for distribution to another system. This is set to YES only when the System Manager for IBM i licensed program is on the system and the product is supported. The PTF_SAVE_FILE column must have a value of YES before using the value in this column. **NO** The PTF save file cannot be distributed. **YES** The PTF save file is released and can be distributed to another system.
PTF_MINIMUM_LEVEL	MINLVL	VARCHAR(2) Nullable	The indicator of the lowest level of the product to which this PTF can be applied. The level can be AA to 99. Contains the null value if the product does not have a level.
PTF_MAXIMUM_LEVEL	MAXLVL	VARCHAR(2) Nullable	The indicator of the highest level of the product to which this PTF can be applied. The level can be AA to 99. Contains the null value if the product does not have a level.
PTF_STATUS_TIMESTAMP	STATTIME	TIMESTAMP Nullable	The date and time that the PTF status was last changed. Contains the null value when the status date and time is not available.
PTF_SUPERCEDED_BY_PTF	SUPERCEDE	VARCHAR(7) Nullable	The identifier of the PTF that has replaced this PTF. This field will be blank when the PTF is not superseded or when the superseding PTF has not been loaded on the system.
PTF_CREATION_TIMESTAMP	CRTTIME	TIMESTAMP Nullable	The date and time that the PTF was created. Contains the null value when the creation date and time cannot be determined.
PTF_TECHNOLOGY_REFRESH_PTF	TRPTF	VARCHAR(3) Nullable	Indicates whether this is a technology refresh PTF. **NO** This is not a technology refresh PTF. **YES** This is a technology refresh PTF.

Examples

- Find which PTFs will be impacted by the next IPL.

```
SELECT PTF_IDENTIFIER, PTF_IPL_ACTION, A.*
  FROM QSYS2.PTF_INFO A
  WHERE PTF_IPL_ACTION <> 'NONE‘
```

- Find which PTFs are loaded but not applied.

```
SELECT PTF_IDENTIFIER, PTF_PRODUCT_DESCRIPTION, A.*
  FROM QSYS2.PTF_INFO A
  WHERE PTF_LOADED_STATUS = 'LOADED'
  ORDER BY PTF_PRODUCT_ID
```

Security Services

These views, procedures, and functions provide security information.

AUTHORIZATION_LIST_INFO view

The AUTHORIZATION_LIST_INFO view returns a list of all objects secured by an authorization list.

The information returned is similar to the information available through the Display Authorization List Objects (DSPAUTLOBJ) CL command and the List Objects Secured by Authorization List (QSYLATLO) API.

Authorization: Detail is returned when one of the following is true:

- The user has *READ authority to the authorization list.
- The user is authorized to the Database Security Administrator function of the IBM i. The Change Function Usage (CHGFCNUSG) command, with a function ID of QIBM_DB_SECADM, can be used to change the list of users allowed to use the function.
- The user has *ALLOBJ special authority.

The following table describes the columns in the view. The system name is AUTHL_INFO. The schema is QSYS2.

Table 95. AUTHORIZATION_LIST_INFO view

Column name	System column name	Data type	Description
AUTHORIZATION_LIST	AUTH_LIST	VARCHAR(10)	The authorization list for this object.
SYSTEM_OBJECT_SCHEMA	SYS_DNAME	VARCHAR(10) Nullable	The library that contains the object. Returns the null value if the object is not in the QSYS or QDLS file system.
SYSTEM_OBJECT_NAME	SYS_ONAME	VARCHAR(10) Nullable	The object that is secured by the authorization list. Returns the null value if the object is not in the QSYS or QDLS file system.
SYSTEM_OBJECT_TYPE	SYS_OTYPE	VARCHAR(8)	The system object type of the secured object.
OBJECT_ATTRIBUTE	OBJATTR	VARCHAR(5) Nullable	The attribute for the secured object's type. Returns the null value if the object has no attribute or if it is not in the QSYS or QDLS file system.
OBJECT_SCHEMA	OSCHEMA	VARCHAR(128) Nullable	The SQL schema name for this object.
OBJECT_NAME	ONAME	VARCHAR(128) Nullable	The SQL name of the object. For an external procedure or an external function, the name will be returned when a single procedure or function exists for that *PGM or *SRVPGM object. Contains the null value if an SQL name could not be returned.

Table 95. AUTHORIZATION_LIST_INFO view (continued)

Column name	System column name	Data type	Description
OBJECT_TYPE	OTYPE	VARCHAR(9) Nullable	The SQL object type. The following values can be returned. **ALIAS** The object is an SQL alias. **FUNCTION** The object is an SQL function. **INDEX** The object is an SQL index. **PACKAGE** The object is an SQL package. **PROCEDURE** The object is an SQL procedure. **ROUTINE** The object is used in SQL by one or more external functions and/or external procedures. **SEQUENCE** The object is an SQL sequence. **TABLE** The object is an SQL table. **TRIGGER** The object is an SQL trigger. **TYPE** The object is an SQL type. **VARIABLE** The object is an SQL global variable. **VIEW** The object is an SQL view. **XSR** The object is an XML schema repository object. Returns the null value if the object is not an SQL object.
OBJECT_OWNER	OWNER	VARCHAR(10)	The owner of the object.
PRIMARY_GROUP	GROUP	VARCHAR(10) Nullable	The user who is the primary group for the object. Returns the null value if there is no primary group for the object.
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	The descriptive text for the secured object. Returns the null value if the object is not in the QSYS or QDLS file system.
ASPGRP	ASPGRP	VARCHAR(10)	The name of the ASP device containing the object. A value of *SYSBAS indicates the system ASP and all basic user ASPs.
AUTHORITY_HOLDER	AUT_HOLDER	VARCHAR(3)	Indicates whether the object is an authority holder. **NO** The object is not an authority holder. **YES** The object is an authority holder.
PATH_NAME	PATH_NAME	DBCLOB(16M) CCSID 1200 Nullable	The path name for the object that is secured by the authorization list. Returns the null value if the object is in the QSYS or QDLS file system.
DLO_NAME	DLO_NAME	VARCHAR(12) Nullable	The document library object (DLO) name for the object. Returns the null value if OBJECT_TYPE is not *DOC (document) or *FLR (folder).
FOLDER_PATH	FOLDER	VARCHAR(63) Nullable	The name of the folder that contains the DLO object. Returns the null value if the object is not in a folder.

Example

Return information about all the object secured by authorization list APP1.

```
SELECT * FROM QSYS2.AUTHORIZATION_LIST_INFO WHERE AUTHORIZATION_LIST = 'APP1';
```

AUTHORIZATION_LIST_USER_INFO view

The AUTHORIZATION_LIST_USER_INFO view returns a list of all authorization lists and their authorities.

The information returned is similar to the information available through the Display Authorization List (DSPAUTL) CL command.

Authorization: None required.

The following table describes the columns in the view. The system name is AUTL_USERS. The schema is QSYS2.

Table 96. AUTHORIZATION_LIST_USER_INFO view

Column name	System column name	Data type	Description
AUTHORIZATION_LIST	AUTL	VARCHAR(10)	The name of the authorization list.
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10)	User profile name. Can contain the following special value. ***PUBLIC** This row contains the public authority for the object.
OBJECT_AUTHORITY	OBJ_AUTH	VARCHAR(12)	The authority that the user has to the object. Contains one of the following values: ***ALL** Allows all operations on the object except those that are limited to the owner or controlled by authorization list management authority. ***CHANGE** Allows all operations on the object except those that are limited to the owner or controlled by object existence authority, object alter authority, object reference authority, and object management authority. ***EXCLUDE** All operations on the object are prohibited. ***USE** Allows access to the object attributes and use of the object. The user cannot change the object. **USER DEFINED** The specific object authorities and data authorities do not match any of the predefined object authority levels.
AUTHORIZATION_LIST_MANAGEMENT	AUTL_MGMT	VARCHAR(3)	The authorization list management authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OWNER	OWNER	VARCHAR(10)	The owner of the authorization list.
OBJECT_OPERATIONAL	OBJOPER	VARCHAR(3)	The object operational authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.

Table 96. AUTHORIZATION_LIST_USER_INFO view (continued)

Column name	System column name	Data type	Description
OBJECT_MANAGEMENT	OBJMGT	VARCHAR(3)	The object management authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OBJECT_EXISTENCE	OBJEXIST	VARCHAR(3)	The object existence authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OBJECT_ALTER	OBJALTER	VARCHAR(3)	The object alter authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OBJECT_REFERENCE	OBJREF	VARCHAR(3)	The object reference authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_READ	DATA_READ	VARCHAR(3)	The data read authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_ADD	DATA_ADD	VARCHAR(3)	The data add authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_UPDATE	DATA_UPD	VARCHAR(3)	The data update authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_DELETE	DATA_DEL	VARCHAR(3)	The data delete authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_EXECUTE	DATA_EXEC	VARCHAR(3)	The data execute authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	The descriptive text for the authorization list. Contains null if the authorization list has no text description.

Example

List the public security settings for all authorization lists.

```
SELECT *
   FROM QSYS2.AUTHORIZATION_LIST_USER_INFO
   WHERE AUTHORIZATION_NAME = '*PUBLIC';
```

DRDA_AUTHENTICATION_ENTRY_INFO view

The DRDA_AUTHENTICATION_ENTRY_INFO view returns user server authentication entry information.

A server authentication entry defines a userid and password to send on a connect request over TCP/IP. A server authentication list is associated with every user profile on the system. The Add Server Authentication Entry (ADDSVRAUTE) command is used to add entries.

When a DRDA connection over TCP/IP is attempted without specifying a userid and password, and password authentication is required, the DB2 for i client checks the server authentication list for the user profile under which the client job is running. If it finds a match between the RDB name on the CONNECT statement and the server name in an authentication entry, or the server name is the special value QDDMDRDASERVER, the associated userid (and password if one exists) is used for the connection.

A server authentication entry can also be used to specify a userid and password to be used for a DDM connection over TCP/IP. When a DDM connection is attempted over TCP/IP, and password authentication is required, the DB2 for i client checks the server authentication list for the user profile under which the client job is running. If it finds a match between the RDB name specified in the DDM file and the server name in an authentication entry, or the server name is the special value QDDMDRDASERVER, the associated userid (and password if one exists) is used for the connection. If no RDB name is specified in the DDM file and the server name is either of the special values QDDMDRDASERVER or QDDMSERVER, the associated userid (and password if one exists) is used for the connection.

Only rows where the AUTHORIZATION_NAME is for a *USRPRF object that you have *OBJOPR and *READ authority to will be returned.

The following table describes the columns in the view. The schema is QSYS2.

Table 97. DRDA_AUTHENTICATION_ENTRY_INFO view

Column Name	System Column Name	Data Type	Description
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10)	The user profile on the client system.
SERVER_NAME	SRVR_NAME	VARGRAPHIC(200) CCSID 1200	The target system for the authentication entry. This is the name of the RDB or QDDMDRDASERVER that is used for connections made on behalf of RDB DDM files or DRDA connections. For a non-RDB DDM file that does not use the RDB directory, the value will be QDDMDRDASERVER or QDDMSERVER. See Client security in a TCP/IP network for more information.
SERVER_AUTHORIZATION_NAME	SRVR_USER	VARGRAPHIC(1000) CCSID 1200	The user profile on the target system.
PASSWORD_STORED	PW_STORED	VARCHAR(3)	Indicates whether a password is stored for the authentication entry. **YES** A password is stored for the authentication entry. **NO** A password is not stored for the authentication entry.

Example

For an auditor, generate a list of user profiles that have authentication entries on the system:

```
SELECT DISTINCT(AUTHORIZATION_NAME)
  FROM QSYS2.DRDA_AUTHENTICATION_ENTRY_INFO
```

FUNCTION_INFO view

The FUNCTION_INFO view contains details about function usage identifiers.

The following table describes the columns in the view. The schema is QSYS2.

Table 98. FUNCTION_INFO view

Column Name	System Column Name	Data Type	Description
FUNCTION_ID	FCNID	VARCHAR(30) Nullable	The function ID.
FUNCTION_CATEGORY	FCNCAT	VARCHAR(10) Nullable	Indicates whether the function is a client or host function. **1 - CLIENT** The function is a locally managed client function within IBM i Navigator. **2 - CLIENT** The function is a locally managed client function, not within IBM i Navigator. **3 - HOST** The function is a host function. **4 - CLIENT** The function is a centrally managed client function within IBM i Navigator. **5 - CLIENT** The function is a centrally managed client function, not within IBM i Navigator.
FUNCTION_TYPE	FCNTYP	VARCHAR(13) Nullable	The type of function. **PRODUCT** The function is a function product. **GROUP** The function is a function group. **ADMINISTRABLE** The function is an administrable function.
FUNCTION_NAME_MESSAGE_TEXT	FCNMSGTXT	VARGRAPHIC(330) CCSID(1200) Nullable	The first-level text for the function-name message ID.
FUNCTION_NAME	FCNNAM	VARGRAPHIC(330) CCSID(1200) Nullable	The text for the function name.
FUNCTION_DESCRIPTION_MESSAGE_TEXT	FCNDESCTXT	VARGRAPHIC(330) CCSID(1200) Nullable	The first-level text for the function-description message ID.
FUNCTION_DESCRIPTION	FCNDESC	VARGRAPHIC(330) CCSID(1200) Nullable	The text for the function description.
FUNCTION_PRODUCT_ID	FCNPRDID	VARCHAR(30) Nullable	The ID of the product that the function is registered for.
FUNCTION_GROUP_ID	FCNGRPID	VARCHAR(30) Nullable	The ID of the function group that the function is grouped with. If the function is not grouped with a function group, this field is set to *NONE.
DEFAULT_USAGE	DFTUG	VARCHAR(7) Nullable	The default usage for the function. **DENIED** The default usage does not allow usage of the function. **ALLOWED** The default usage allows usage of the function.
ALLOBJ_INDICATOR	ALLOBJ	VARCHAR(8) Nullable	Indicates whether a user with *ALLOBJ special authority can use the function. **NOT USED** The user, its groups, or default must allow usage of the function. **USED** A user with *ALLOBJ special authority is always allowed to use the function.
USAGE_INFORMATION_INDICATOR	USGINFO	VARCHAR(3) Nullable	Indicates whether there is usage information defined for the function. **NO** There is no usage information defined for the function. **YES** There is usage information defined for the function.

Example

Determine what function usage IDs exist and their default configuration.

```
SELECT * FROM QSYS2.FUNCTION_INFO ORDER BY FUNCTION_ID
```

FUNCTION_USAGE view

The FUNCTION_USAGE view contains function usage configuration details.

Only users with *SECADM user special authority can examine the function usage configuration details returned with this view. Users without *SECADM authority who attempt to reference this view will get SQLCODE -443.

The following table describes the columns in the view. The schema is QSYS2.

Table 99. FUNCTION_USAGE view

Column Name	System Column Name	Data Type	Description
FUNCTION_ID	FCNID	VARCHAR(30)	The ID of the function.
USER_NAME	USER_NAME	VARCHAR(10)	The name of the user profile that has a usage setting for this function
USAGE	USAGE	VARCHAR(7)	Usage setting. **ALLOWED** The user profile is allowed to use the function. **DENIED** The user profile is not allowed to use the function.
USER_TYPE	USER_TYPE	VARCHAR(5)	Type of user profile. **USER** The user profile is a user. **GROUP** The user profile is a group.

Example

Determine what function usage has been granted or revoked.

```
SELECT * FROM QSYS2.FUNCTION_USAGE ORDER BY FUNCTION_ID, USER_NAME
```

GROUP_PROFILE_ENTRIES view

The GROUP_PROFILE_ENTRIES view contains one row for each user profile that is part of a group profile.

Both group profile (GRPPRF) and supplemental group profile (SUPGRPPRF) information is considered for each user profile.

The following table describes the columns in the view. The schema is QSYS2.

Table 100. GROUP_PROFILE_ENTRIES view

Column Name	System Column Name	Data Type	Description
GROUP_PROFILE_NAME	GROUPNAME	VARCHAR(128)	Group profile name
USER_PROFILE_NAME	USERNAME	VARCHAR(128)	User profile name
USER_TEXT	USER_TEXT	VARCHAR(50) Nullable	User profile text description.

OBJECT_PRIVILEGES view

The OBJECT_PRIVILEGES view returns a row for every user authorized to an object, along with their associated object and data authorities.

The information returned is similar to the information available through the Display Object Authority (DSPOBJAUT) CL command.

Authorization: All authorized users are returned for an object when at least one of the following is true:

- The caller has *OBJMGT authority.
- The caller is the owner of the object.
- The object is an authorization list.
- The caller is authorized to the Database Security Administrator function of IBM i. The Change Function Usage (CHGFCNUSG) command, with a function ID of QIBM_DB_SECADM, can be used to change the list of users allowed to use the function.

Otherwise, only authorizations for the caller are returned.

The following table describes the columns in the view. The system name is OBJ_PRIV. The schema is QSYS2.

Table 101. OBJECT_PRIVILEGES view

Column name	System column name	Data type	Description
OBJECT_SCHEMA	OSCHEMA	VARCHAR(128)	The SQL schema name for this object.
OBJECT_NAME	NAME	VARCHAR(128) Nullable	The SQL name of the object. For an external procedure or an external function, the name will be returned when a single procedure or function exists for that *PGM or *SRVPGM object. Contains the null value if an SQL name could not be returned.
SYSTEM_OBJECT_SCHEMA	SYS_DNAME	VARCHAR(10)	The library that contains the object.
SYSTEM_OBJECT_NAME	SYS_ONAME	VARCHAR(10)	The system object name.
OBJECT_TYPE	OBJTYPE	VARCHAR(8)	The system object type.
SQL_OBJECT_TYPE	SQLTYPE	VARCHAR(9) Nullable	The SQL object type. The following values can be returned. **ALIAS** The object is an SQL alias. **FUNCTION** The object is an SQL function. **INDEX** The object is an SQL index. **PACKAGE** The object is an SQL package. **PROCEDURE** The object is an SQL procedure. **ROUTINE** The object is used in SQL by one or more external functions and/or external procedures. **SEQUENCE** The object is an SQL sequence. **TABLE** The object is an SQL table. **TRIGGER** The object is an SQL trigger. **TYPE** The object is an SQL type. **VARIABLE** The object is an SQL global variable. **VIEW** The object is an SQL view. **XSR** The object is an XML schema repository object. Returns the null value if the object is not an SQL object.

Table 101. OBJECT_PRIVILEGES view (continued)

Column name	System column name	Data type	Description
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10)	User profile name. Can contain the following special value. ***PUBLIC** This row contains the public authority for the object.
OBJECT_AUTHORITY	OBJ_AUTH	VARCHAR(12)	The authority that the user has to the object. Contains one of the following special values: ***ALL** Allows all operations on the object except those that are limited to the owner or controlled by authorization list management authority. ***AUTL** The public authority specified in the authorization list used by this object is used ***CHANGE** Allows all operations on the object except those that are limited to the owner or controlled by object existence authority, object alter authority, object reference authority, and object management authority. ***EXCLUDE** All operations on the object are prohibited. ***USE** Allows access to the object attributes and use of the object. The user cannot change the object. **USER DEFINED** The specific object authorities and data authorities do not match any of the predefined object authority levels.
OWNER	OWNER	VARCHAR(10)	The user profile that owns the object.
OBJECT_OPERATIONAL	OBJOPER	VARCHAR(3)	Indicates the object operational authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OBJECT_MANAGEMENT	OBJMGT	VARCHAR(3)	The object management authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OBJECT_EXISTENCE	OBJEXIST	VARCHAR(3)	The object existence authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
OBJECT_ALTER	OBJALTER	VARCHAR(3)	The object alter authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.

Table 101. OBJECT_PRIVILEGES view (continued)

Column name	System column name	Data type	Description
OBJECT_REFERENCE	OBJREF	VARCHAR(3)	The object reference authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_READ	DATA_READ	VARCHAR(3)	The data read authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_ADD	DATA_ADD	VARCHAR(3)	The data add authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_UPDATE	DATA_UPD	VARCHAR(3)	The data update authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_DELETE	DATA_DEL	VARCHAR(3)	The data delete authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
DATA_EXECUTE	DATA_EXEC	VARCHAR(3)	The data execute authority for AUTHORIZATION_NAME. **NO** The user does not have this authority. **YES** The user has this authority.
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	The descriptive text for this object. Contains null if the object has no text description.

Example

Find user profiles that are publicly accessible.

```
SELECT *
   FROM QSYS2.OBJECT_PRIVILEGES
   WHERE SYSTEM_OBJECT_SCHEMA = 'QSYS' AND
         OBJECT_TYPE = '*USRPRF' AND
         AUTHORIZATION_NAME = '*PUBLIC' AND
         OBJECT_AUTHORITY <> '*EXCLUDE';
```

SET_COLUMN_ATTRIBUTE procedure

The SET_COLUMN_ATTRIBUTE procedure sets the SECURE attribute for a column so variable values used for the column cannot be seen in the database monitor or plan cache.

SET_COLUMN_ATTRIBUTE (*schema-name* , *table-name* , *column-name* , *attribute*)

The schema is SYSPROC.

schema-name
A character string expression containing the system name of a schema.

table-name
A character string expression containing the system name of a table.

column-name
A character string expression containing the system name of a column.

attribute
A character string expression containing the attribute to set for the column.

Valid values are:

SECURE NO
This column does not contain data that needs to be secured in a database monitor or plan cache

SECURE YES
This column contains data that needs to be secured in a database monitor or plan cache.

All variable values for any query that references this column will not be visible in a database monitor or plan cache unless the security officer has started the database monitor or the security officer is accessing the plan cache. All host variable values will appear as *SECURE when examined from the monitor and plan cache unless the user is the QSECOFR user.

The secure setting for a column is shown in the SECURE column of the QSYS2/SYSCOLUMNS2 catalog.

Example

Set the credit card column in the ORDERS table so it is secure.

```
CALL SYSPROC.SET_COLUMN_ATTRIBUTE('LIB1', 'ORDERS', 'CCNBR', 'SECURE YES');
```

SQL_CHECK_AUTHORITY scalar function

The SQL_CHECK_AUTHORITY scalar function returns an indication of whether the user is authorized to query the specified *FILE object.

```
►► SQL_CHECK_AUTHORITY ── ( ── library-name ── , ── file-name ── ) ►◄
```

The schema is QSYS2.

library-name
Library name containing the file.

file-name
File name for which authority will be examined.

The result of the function is a SMALLINT.

The returned value is:

0
If the user does not have authority to query the file, the object is not a *FILE object, or the object does not exist.

1
If the user is authorized to query the file.

USER_INFO view

The USER_INFO view contains information about user profiles.

The values returned for the columns in the view are closely related to the values returned by Retrieve User Information (QSYRUSRI) API. Refer to the API for more detailed information.

Authorization: Only *USRPRF objects that the user has *OBJOPR and *READ authority to will be returned. To see a non-null value for the USER_DEFAULT_PASSWORD column, the user must have *ALLOBJ and *SECADM authority.

The following table describes the columns in the view. The schema is QSYS2.

Table 102. USER_INFO view

Column Name	System Column Name	Data Type	Description
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10) Nullable	User profile name.
PREVIOUS_SIGNON	PRVSIGNON	TIMESTAMP Nullable	The date and time the user last signed on.
SIGN_ON_ATTEMPTS_NOT_VALID	SIGNONINV	INTEGER Nullable	The number of sign-on attempts that were not valid since the last successful sign-on.
STATUS	STATUS	VARCHAR(10) Nullable	The status of the user profile.
PASSWORD_CHANGE_DATE	PWDCHGDAT	TIMESTAMP Nullable	The date the user's password was last changed.
NO_PASSWORD_INDICATOR	NOPWD	VARCHAR(3) Nullable	Indicates whether *NONE is specified for the password in the user profile. **NO** The password in the user profile is not *NONE. **YES** The password in the user profile is *NONE.
PASSWORD_EXPIRATION_INTERVAL	PWDEXPITV	INTEGER Nullable	The number of days (from 1 through 366) the user's password can remain active before it must be changed.
DATE_PASSWORD_EXPIRES	PWDEXPDAT	TIMESTAMP Nullable	The date the user's password expires.
DAYS_UNTIL_PASSWORD_EXPIRES	PWDDAYSEXP	INTEGER Nullable	The number of days until the password will expire. Contains null if the password will not expire within the number of days specified by the password expiration warning (QPWDEXPWRN) system value.
SET_PASSWORD_TO_EXPIRE	PWDEXP	VARCHAR(3) Nullable	Indicates whether the user's password is set to expire, requiring the user to change the password when signing on.
USER_CLASS_NAME	USRCLS	VARCHAR(10) Nullable	The user's class name.
SPECIAL_AUTHORITIES	SPCAUT	VARCHAR(88) Nullable	A list of the special authorities the user has.
GROUP_PROFILE_NAME	GRPPRF	VARCHAR(10) Nullable	The name of the group profile.
SUPPLEMENTAL_GROUP_COUNT	SUPGRPCNT	SMALLINT	The number of supplemental groups in the SUPPLEMENTAL_GROUP_LIST column.
SUPPLEMENTAL_GROUP_LIST	SUPGRPLIST	VARCHAR(150) Nullable	A list of supplemental groups for the user profile. Up to 15 supplemental groups are returned. Each entry except for the last one is padded with blanks to fill 10 characters. Contains null if the user has no supplemental groups.
OWNER	OWNER	VARCHAR(10) Nullable	This field indicates who is to own objects created by this user.

Table 102. USER_INFO view (continued)

Column Name	System Column Name	Data Type	Description
GROUP_AUTHORITY	GRPAUT	VARCHAR(10) Nullable	The authority the user's group profile has to objects the user creates.
ASSISTANCE_LEVEL	ASTLVL	VARCHAR(10) Nullable	The user interface that the user will use.
CURRENT_LIBRARY_NAME	CURLIB	VARCHAR(10) Nullable	The name of the user's current library.
INITIAL_MENU_NAME	INLMNU	VARCHAR(10) Nullable	The initial menu for the user.
INITIAL_MENU_LIBRARY_NAME	INLMNULIB	VARCHAR(10) Nullable	The name of the library that the initial menu is in.
INITIAL_PROGRAM_NAME	INITPGM	VARCHAR(10) Nullable	The initial program for the user.
INITIAL_PROGRAM_LIBRARY_NAME	INITPGMLIB	VARCHAR(10) Nullable	The name of the library that the initial program is in.
LIMIT_CAPABILITIES	LMTCPB	VARCHAR(10) Nullable	Indicates whether the user has limited capabilities.
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	The descriptive text for the user profile.
DISPLAY_SIGNON_INFORMATION	DSPSGNINF	VARCHAR(10) Nullable	Indicates whether the sign-on information display is shown when the user signs on.
LIMIT_DEVICE_SESSIONS	LMTDEVSSN	VARCHAR(10) Nullable	Specifies if the number of device sessions allowed for a user is limited.
KEYBOARD_BUFFERING	KBDBUF	VARCHAR(10) Nullable	The keyboard buffering value that is used when a job is initialized for this user.
MAXIMUM_ALLOWED_STORAGE	MAXSTGLRG	BIGINT Nullable	The maximum amount of auxiliary storage (in kilobytes) that can be assigned to store permanent objects owned by the user. Contains null if the user has no maximum storage.
STORAGE_USED	STGUSED	BIGINT Nullable	The amount of auxiliary storage (in kilobytes) occupied by this user's owned objects on *SYSBAS. The QSYS2.USER_STORAGE catalog should be used to determine the storage consumed on all ASPs.
HIGHEST_SCHEDULING_PRIORITY	PTYLMT	CHAR(1) Nullable	The highest scheduling priority the user is allowed to have for each job submitted to the system.
JOB_DESCRIPTION_NAME	JOBD	VARCHAR(10) Nullable	The name of the job description used for jobs that start through subsystem work station entries.
JOB_DESCRIPTION_LIBRARY_NAME	JOBDLIB	VARCHAR(10) Nullable	Job description library name.
ACCOUNTING_CODE	ACGCDE	VARCHAR(15) Nullable	The accounting code that is associated with this user.
MESSAGE_QUEUE_NAME	MSGQ	VARCHAR(10) Nullable	The name of the message queue that is used by this user.
MESSAGE_QUEUE_LIBRARY_NAME	MSGQLIB	VARCHAR(10) Nullable	The name of the library the message queue is in.

Table 102. USER_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MESSAGE_QUEUE_DELIVERY_METHOD	DLVRY	VARCHAR(10) Nullable	How the messages are delivered to the message queue used by the user.
MESSAGE_QUEUE_SEVERITY	SEV	SMALLINT Nullable	The lowest severity that a message can have and still be delivered to a user in break or notify mode
OUTPUT_QUEUE_NAME	OUTQ	VARCHAR(10) Nullable	The output queue used by this user.
OUTPUT_QUEUE_LIBRARY_NAME	OUTQLIB	VARCHAR(10) Nullable	The name of the library where the output queue is located.
PRINT_DEVICE	PRTDEV	VARCHAR(10) Nullable	The printer used to print for this user.
SPECIAL_ENVIRONMENT	SPCENV	VARCHAR(10) Nullable	The special environment the user operates in after signing on.
ATTENTION_KEY_HANDLING_ PROGRAM_NAME	ATNPGM	VARCHAR(10) Nullable	The attention key handling program for this user.
ATTENTION_KEY_HANDLING_ PROGRAM_LIBRARY_NAME	ATNPGMLIB	VARCHAR(10) Nullable	The name of the library where the program is located.
LANGUAGE_ID	LANGID	VARCHAR(10) Nullable	The language ID used by the system for this user.
COUNTRY_OR_REGION_ID	CNTRYID	VARCHAR(10) Nullable	Country or region ID.
CHARACTER_CODE_SET_ID	CCSID	VARCHAR(6) Nullable	The CCSID for the user.
USER_OPTIONS	USROPT	VARCHAR(77) Nullable	A list of the options for users to customize their environment. Contains null if there are no user options.
SORT_SEQUENCE_TABLE_NAME	SRTSEQ	VARCHAR(10) Nullable	The name of the sort sequence table used for string comparisons.
SORT_SEQUENCE_TABLE_LIBRARY_NAME	SRTSEQLIB	VARCHAR(10) Nullable	The name of the library that is used to locate the sort sequence table.
OBJECT_AUDITING_VALUE	OBJAUD	VARCHAR(10) Nullable	The object auditing value for this user.
USER_ACTION_AUDIT_LEVEL	AUDLVL	VARCHAR(341) Nullable	The action audit values for this user.
GROUP_AUTHORITY_TYPE	GRPAUTTYP	VARCHAR(10) Nullable	The type of authority the user's group profile has to objects the user creates.
USER_ID_NUMBER	UID	BIGINT Nullable	The user ID number for the user profile.
GROUP_ID_NUMBER	GID	BIGINT Nullable	The group ID number for the user profi
LOCALE_JOB_ATTRIBUTES	SETOBJATR	VARCHAR(88) Nullable	A list of the job attributes that are taken from the user's locale path.

Table 102. USER_INFO view (continued)

Column Name	System Column Name	Data Type	Description
GROUP_MEMBER_INDICATOR	GRPMBR	VARCHAR(3) Nullable	Whether this user is a group that has members.
DIGITAL_CERTIFICATE_INDICATOR	DCIND	VARCHAR(3) Nullable	Whether there are digital certificates associated with this user.
CHARACTER_IDENTIFIER_CONTROL	CHRIDCTL	VARCHAR(10) Nullable	The character identifier control for the user.
LOCAL_PASSWORD_MANAGEMENT	LCLPWDMGT	VARCHAR(3) Nullable	Indicates if password is managed locally.
BLOCK_PASSWORD_CHANGE	PWDCHGBLK	VARCHAR(10) Nullable	Specifies the time period, in hours, during which a password is blocked from being changed following the prior successful password change operation.
USER_ENTITLEMENT_REQUIRED	ENTITLERQD	VARCHAR(3) Nullable	Whether a user entitlement is required for this user profile.
USER_EXPIRATION_INTERVAL	USREXPITV	SMALLINT Nullable	The number of days (from 1 through 366) before the user profile is automatically disabled.
USER_EXPIRATION_DATE	ESREXPDATE	TIMESTAMP Nullable	The date when the user profile expires and is automatically disabled or deleted.
USER_EXPIRATION_ACTION	ACTION	VARCHAR(8) Nullable	The action that will occur when the user profile has expired.
HOME_DIRECTORY	HOMEDIR	VARGRAPHIC(1024) CCSID 1200 Nullable	The home directory for this user profile.
LOCALE_PATH_NAME	LOCALE	VARGRAPHIC(1024) CCSID 1200 Nullable	The locale path name that is assigned to the user profile when a job is started.
USER_DEFAULT_PASSWORD	DFTPWD	VARCHAR(3) Nullable	The password is the default password. **NO** The password is not the default password. **YES** The password appears to be the default password since it matches the user profile name. Contains null if not authorized to view this information.
USER_OWNER	USER_OWNER	VARCHAR(10) Nullable	The user profile that owns this user profile.
USER_CREATOR	CREATOR	VARCHAR(10) Nullable	The user profile that created this user profile.
SIZE	SIZE	DECIMAL(15,0) Nullable	Size of the user profile, in bytes.
CREATION_TIMESTAMP	TIMESTAMP	TIMESTAMP Nullable	Timestamp of when the user profile was created.
LAST_USED_TIMESTAMP	LASTUSED	TIMESTAMP Nullable	The date the user profile was used last. The time portion of the timestamp will always be 0.

Table 102. USER_INFO view (continued)

Column Name	System Column Name	Data Type	Description
DAYS_USED_COUNT	DAYSUSED	INTEGER Nullable	The number of days the user profile has been used on the system.
LAST_RESET_TIMESTAMP	LASTRESET	TIMESTAMP Nullable	The date when the days used count was last reset to zero. The time portion of the timestamp will always be 0.
PASE_SHELL_PATH	SHELL_PATH	VARCHAR(1024) CCSID 1208 Nullable	Path to the user's PASE shell. If AUTHORIZATION_NAME is QSYS, this column contains the default shell path used for all user profiles that have not had a value explicitly set. Returns the null value if a value has not been set using the QSYS2.SET_PASE_SHELL_INFO procedure.

Example

Determine which users are having trouble signing on.

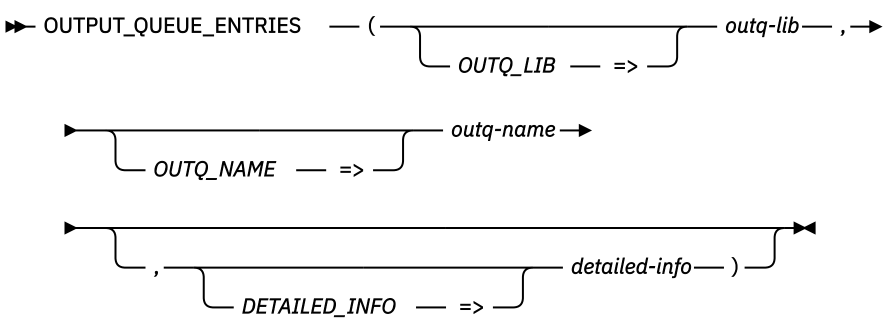
```
SELECT * FROM QSYS2.USER_INFO
  WHERE SIGN_ON_ATTEMPTS_NOT_VALID > 0
```

Spool Services

This view and function provide information about spooled files.

OUTPUT_QUEUE_ENTRIES table function

The OUTPUT_QUEUE_ENTRIES table function returns one row for each spooled file in an output queue.



The schema is QSYS2.

To invoke this function, the caller must have:

- Read authority to the output queue object, or
- *JOBCTL special authority and the output queue has OPRCTL(*YES), or
- *SPLCTL special authority

outq-lib
A character or graphic string expression that identifies the name of the library containing *outq-name*. If this parameter is blank, the default of *LIBL is used.

outq-name
A character or graphic string expression that identifies the name of an output queue.

detailed-info
A character or graphic string expression that indicates the type of information to be returned.

***YES**
All the information available for the output queue is returned.

***NO**
Only the general information is returned for the output queue. This is the information in the columns prior to the ACCOUNTING_CODE column. This is the default.

The result of the function is a table containing rows with the format shown in the following table. All the columns are nullable.

Table 103. OUTPUT_QUEUE_ENTRIES table function

Column Name	Data Type	Description
CREATE_TIMESTAMP	TIMESTAMP	The timestamp when the file was created.
SPOOLED_FILE_NAME	VARCHAR(10)	The file name that was specified by the user program when the file was created, or the name of the device file used to create this file.
USER_NAME	VARCHAR(10)	The name of the user profile that produced the file.
USER_DATA	VARCHAR(10)	The user-specified data that describes this file. Contains null if there is no user-specified data.
STATUS	VARCHAR(15)	Status of the spooled file. **CLOSED** The file has been completely processed by a program but SCHEDULE(*JOBEND) was specified and the job that produced the file has not yet finished. **DEFERRED** Printing of the file has been deferred. **DELETED** The file has been deleted. **HELD** The file has been held. **MESSAGE WAITING** This file has a message which needs a reply or an action to be taken. **OPEN** The file has not been completely processed and is not ready to be selected by a writer. **PENDING** The file is pending to be printed. **PRINTING** The file has been completely sent to the printer but print complete status has not been sent back. **READY** The file is available to be written. **SAVED** The file has been printed and then saved. This file remains saved until it is released. **SENDING** The file is being sent or has been sent to a remote system. **WRITING** This file is currently being produced by the writer.
SIZE	INTEGER	The size of the spooled file, in kilobytes.
TOTAL_PAGES	INTEGER	The total number of pages in the file.
COPIES	SMALLINT	The number of copies remaining to print.
FORM_TYPE	VARCHAR(10)	The type of form that should be loaded in the printer to print this file.
JOB_NAME	VARCHAR(28)	The qualified job name that produced the file.

Table 103. OUTPUT_QUEUE_ENTRIES table function (continued)

Column Name	Data Type	Description
DEVICE_TYPE	VARCHAR(10)	The type of data stream used to represent the file. ***AFPDS** Advanced Function Presentation data stream ***AFPDSLINE** AFPDS data mixed with 1403 line data ***IPDS** Intelligent printer data stream ***LINE** 1403 line data ***SCS** Systems Network Architecture (SNA) character stream ***USERASCII** ASCII data
OUTPUT_PRIORITY	SMALLINT	The priority of the spooled file.
FILE_NUMBER	INTEGER	The spooled file number of the specified file.
SYSTEM	VARCHAR(8)	The name of the system where the job that created the spooled file ran.
Values for the following columns are returned when the DETAILED_INFO parameter is *YES. Otherwise, the columns will contain the null value.		
ACCOUNTING_CODE	VARCHAR(15)	An identifier assigned by the system to record the resources used to write this file.
EXPIRATION_DATE	DATE	The date the file will be eligible for removal from the system by the Delete Expired Spooled Files (DLTEXPSPLF) command. Contains the null value if the file will not expire.
SAVE_AFTER_WRITE	VARCHAR(4)	Indicates whether this file is to be saved after it is written. ***NO** The file is deleted after it has been written. ***YES** The file is set to save status after it has been written.
PAGE_LENGTH	INTEGER	The page length, in lines per page, used by the spooled file.
LINES_PER_INCH	DECIMAL(5,1)	The number of lines per vertical inch defined in the printer file.
PAGE_WIDTH	INTEGER	The page width, in characters per printed line, used by the spooled file.
CHARACTERS_PER_INCH	DECIMAL(5,1)	The number of characters per horizontal inch, defined in the printer file.
PRINT_FIDELITY	VARCHAR(8)	The kind of error handling that is performed when printing. ***ABSOLUTE** The file is printed only if it can be printed exactly as specified in the data stream. ***CONTENT** The printing overrides errors in the data stream and continues printing with the printers best quality based on the content fidelity.
PAGE_ROTATION	VARCHAR(5)	The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. ***AUTO** Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality. ***DEVD** The operating system sends a device default rotation value to the printer. Page rotation is dependent on the printer's specifications. ***COR** Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long.

Table 103. OUTPUT_QUEUE_ENTRIES table function (continued)

Column Name	Data Type	Description
PRINT_BOTH_SIDES	VARCHAR(7)	How the information prints. ***FORMDF** The file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, and *AFPDSLINE printer device type files. ***NO** The printing on the page is on one side only. ***YES** The printing is on both sides of the page with the top of each page the same for both sides. ***TUMBLE** The printing is on both sides with the top of one printed page at the opposite end from the top of the other printed page.
FILE_AVAILABLE	VARCHAR(8)	The time when this file becomes available to an output device for processing. ***IMMED** The file is available as soon as the file is opened. ***FILEEND** The file is available as soon as the file is closed. ***JOBEND** The file is available when the job that owns the file is completed.
STARTING_PAGE	VARCHAR(10)	The page at which printing is to start for the file. Can contain the following special value: ***ENDPAGE** Printing starts with the last page.
ENDING_PAGE	VARCHAR(10)	The page at which printing is to end for the file. Can contain the following special value: ***END** Printing ends with the last page.
DEVICE_FILE_LIBRARY	VARCHAR(10)	The name of the library that contains the device file.
DEVICE_FILE_NAME	VARCHAR(10)	The name of the device file used to create the spooled file.
PROGRAM_THAT_OPENED_FILE_LIBRARY	VARCHAR(10)	The name of the library that contains the program that opened the file. Contains null when the program is not known.
PROGRAM_THAT_OPENED_FILE_NAME	VARCHAR(10)	The name of the program that opened the spooled file. Contains null when the program is not known.
FORM_DEFINITION_LIBRARY	VARCHAR(10)	The name of the library that contains the form definition. Contains null if FORM_DEFINITION_NAME is a special value or if no form definition is specified for this spooled file.
FORM_DEFINITION_NAME	VARCHAR(10)	The name of the form definition to use for this print request. Can contain one of the following special values: ***DEVD** The form definition in the printer device description will be used. ***INLINE** The form definition defined in the spooled file data stream will be used. ***INLINED** The form definition defined in the spooled file data stream will be used. If a form definition is not found, the form definition in the printer device description will be used. **F1DFLT** The form definition defined in the spooled file data stream will be used. Contains null when no form definition is specified for this spooled file.
PAGE_DEFINITION_LIBRARY	VARCHAR(10)	The name of the library containing the page definition. Contains the null value for *LINE or *AFPDSLINE printer device type files.
PAGE_DEFINITION_NAME	VARCHAR(10)	The name of the page definition to use for the file. Contains the null value for *LINE or *AFPDSLINE printer device type files.

Table 103. OUTPUT_QUEUE_ENTRIES table function (continued)

Column Name	Data Type	Description
FRONT_OVERLAY_LIBRARY	VARCHAR(10)	The name of the library containing the front overlay. Can contain one of these special values: ***CURLIB** The current ibrary is searched the front overlay. ***LIBL** The library list is used to locate the front overlay. Contains null when FRONT_OVERLAY_NAME is *NONE.
FRONT_OVERLAY_NAME	VARCHAR(10)	The name of the front overlay. Can contain the following special value: ***NONE** The file does not use the front overlay.
BACK_OVERLAY_LIBRARY	VARCHAR(10)	The name of the library containing the back overlay. Contains null when BACK_OVERLAY_NAME is a special value.
BACK_OVERLAY_NAME	VARCHAR(10)	The name of the back overlay. Can contain the following special values: ***FRONTOVL** The back overlay is the same as the front overlay. ***NONE** The file does not use the back overlay.
CHARACTER_SET_LIBRARY	VARCHAR(10)	The name of the library containing the font character set object. Can contain one of these special values: ***CURLIB** The current library is searched for the font character set object. ***LIBL** The library list is used to locate the font character set object. Contains null when CHARACTER_SET_NAME is *FONT.
CHARACTER_SET_NAME	VARCHAR(10)	The name of the font character set object used to print this file. Can contain the following special value: ***FONT** The information specified on the font parameter is used instead of the character set and code page.
CODE_PAGE_LIBRARY	VARCHAR(10)	The name of the library containing the code page used to print this spooled file. Can contain one of these special values: ***CURLIB** The current library is searched for the code page name. ***LIBL** The library list is used to locate the code page name. Contains null when no code page is specified for this spooled file.
CODE_PAGE_NAME	VARCHAR(10)	The name of the code page used to print this spooled file. Contains null when no code page is specified for this spooled file.
CHARACTER_SET_POINTSIZE	DECIMAL(5,1)	The point size in which this file's characters should be printed. Contains null if the character set does not have a point size.
CODED_FONT_LIBRARY	VARCHAR(10)	The name of the library containing the coded font used to print this spooled file. Can contain one of these special values: ***CURLIB** The current library is searched for the coded font. ***LIBL** The library list is used to locate the coded font. Contains null when CODED_FONT_NAME is *FNTCHRSET.
CODED_FONT_NAME	VARCHAR(10)	The name of the coded font used to print this spooled file. Can contain the following special value: ***FNTCHRSET** The values used are the values specified on the character set name and library name and code page name and library name fields.
CODED_FONT_POINTSIZE	DECIMAL(5,1)	The point size in which this file's characters should be printed. Contains null if the coded font does not have a point size.
MULTIBYTE_DATA	VARCHAR(10)	Whether the file can contain double-byte character set (DBCS) data, Unicode data, or both. Values are *YES and *NO.

Table 103. OUTPUT_QUEUE_ENTRIES table function (continued)

Column Name	Data Type	Description
DBCS_CODED_FONT_LIBRARY	VARCHAR(10)	The name of the library containing the DBCS-coded font. Can contain one of these special values: ***CURLIB** The current library is searched for the DBCS-coded font. ***LIBL** The library list is used to locate the DBCS-coded font. Contains null when DBCS_CODED_FONT_NAME is *SYSVAL.
DBCS_CODED_FONT_NAME	VARCHAR(10)	The name of the DBCS-coded font used to print DBCS-coded data on printers configured as AFP(*YES). Can contain the following special value: ***SYSVAL** The DBCS-coded font specified in the system value is used.
DBCS_CODED_FONT_POINTSIZE	DECIMAL(5,1)	The point size in which this file's DCBS characters should be printed. Contains null if the DBCS-coded font does not have a point size.

Example

Find the 100 largest spool files in the QEZJOBLOG output queue. Since no detailed information is needed, specify *NO to avoid the additional processing.

```
SELECT *
  FROM TABLE(QSYS2.OUTPUT_QUEUE_ENTRIES('*LIBL', 'QEZJOBLOG', '*NO')) A
  ORDER BY SIZE DESC
  FETCH FIRST 100 ROWS ONLY
```

OUTPUT_QUEUE_ENTRIES view

The OUTPUT_QUEUE_ENTRIES view returns one row for each spooled file in every output queue.

The schema is QSYS2.

Rows will be returned for spooled files when the caller has:

- Execute authority to the output queue library and
  - Read authority to the output queue object, or
  - *JOBCTL special authority and the output queue has OPRCTL(*YES), or
  - *SPLCTL special authority

To achieve the best performance when querying the OUTPUT_QUEUE_ENTRIES view, the use of a WHERE clause is recommended if you are interested in examining specific output queue libraries or output queues. If the intent of the query is to examine all spool files, consider whether all attributes of the spool file are necessary. If not, a better performing query may be possible using the OUTPUT_QUEUE_ENTRIES table function with DETAIL_INFO set to *NO.

The following table describes the columns in the view. The schema is QSYS2.

Table 104. OUTPUT_QUEUE_ENTRIES view

Column Name	System Column Name	Data Type	Description
OUTPUT_QUEUE_NAME	OUTQ	VARCHAR(10)	Name of the output queue containing the spooled file.
OUTPUT_QUEUE_LIBRARY_NAME	OUTQLIB	VARCHAR(10)	The name of the library that contains the output queue.
CREATE_TIMESTAMP	CREATED	TIMESTAMP	The timestamp when the file was created.
SPOOLED_FILE_NAME	SPOOLNAME	VARCHAR(10)	The file name that was specified by the user program when the file was created, or the name of the device file used to create this file.
USER_NAME	USER_NAME	VARCHAR(10)	The name of the user profile that produced the file.
USER_DATA	USER_DATA	VARCHAR(10) Nullable	The user-specified data that describes this file. Contains null if there is no user-specified data.

Table 104. OUTPUT_QUEUE_ENTRIES view (continued)

Column Name	System Column Name	Data Type	Description
STATUS	STATUS	VARCHAR(15)	Status of the spooled file. **CLOSED** The file has been completely processed by a program but SCHEDULE(*JOBEND) was specified and the job that produced the file has not yet finished. **DEFERRED** Printing of the file has been deferred. **DELETED** The file has been deleted. **HELD** The file has been held. **MESSAGE WAITING** This file has a message which needs a reply or an action to be taken. **OPEN** The file has not been completely processed and is not ready to be selected by a writer. **PENDING** The file is pending to be printed. **PRINTING** The file has been completely sent to the printer but print complete status has not been sent back. **READY** The file is available to be written. **SAVED** The file has been printed and then saved. This file remains saved until it is released. **SENDING** The file is being sent or has been sent to a remote system. **WRITING** This file is currently being produced by the writer.
SIZE	SIZE	INTEGER	The size of the spooled file, in kilobytes.
TOTAL_PAGES	PAGES	INTEGER	The total number of pages in the file.
COPIES	COPIES	SMALLINT	The number of copies remaining to print.
FORM_TYPE	FORM_TYPE	VARCHAR(10)	The type of form that should be loaded in the printer to print this file.
JOB_NAME	JOB_NAME	VARCHAR(28)	The qualified job name that produced the file.
DEVICE_TYPE	DEVTYPE	VARCHAR(10)	The type of data stream used to represent the file. ***AFPDS** Advanced Function Presentation data stream ***AFPDSLINE** AFPDS data mixed with 1403 line data ***IPDS** Intelligent printer data stream ***LINE** 1403 line data ***SCS** Systems Network Architecture (SNA) character stream ***USERASCII** ASCII data
OUTPUT_PRIORITY	OUTPTY	SMALLINT	The priority of the spooled file.
FILE_NUMBER	FILENUM	INTEGER	The spooled file number of the specified file.
SYSTEM	SYSTEM	VARCHAR(8)	The name of the system where the job that created the spooled file ran.
ACCOUNTING_CODE	ACGCDE	VARCHAR(15)	An identifier assigned by the system to record the resources used to write this file.
EXPIRATION_DATE	EXPDATE	DATE Nullable	The date the file will be eligible for removal from the system by the Delete Expired Spooled Files (DLTEXPSPLF) command. Contains the null value if the file will not expire.

Table 104. OUTPUT_QUEUE_ENTRIES view (continued)

Column Name	System Column Name	Data Type	Description
SAVE_AFTER_WRITE	SAVEAFTER	VARCHAR(4)	Indicates whether this file is to be saved after it is written. ***NO** The file is deleted after it has been written. ***YES** The file is set to save status after it has been written.
PAGE_LENGTH	PAGELEN	INTEGER	The page length, in lines per page, used by the spooled file.
LINES_PER_INCH	LPI	DECIMAL(5,1)	The number of lines per vertical inch defined in the printer file.
PAGE_WIDTH	WIDTH	INTEGER	The page width, in characters per printed line, used by the spooled file.
CHARACTERS_PER_INCH	CPI	DECIMAL(5,1)	The number of characters per horizontal inch, defined in the printer file.
PRINT_FIDELITY	FIDELITY	VARCHAR(8)	The kind of error handling that is performed when printing. ***ABSOLUTE** The file is printed only if it can be printed exactly as specified in the data stream. ***CONTENT** The printing overrides errors in the data stream and continues printing with the printers best quality based on the content fidelity.
PAGE_ROTATION	ROTATION	VARCHAR(5)	The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. ***AUTO** Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality. ***DEVD** The operating system sends a device default rotation value to the printer. Page rotation is dependent on the printer's specifications. ***COR** Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long.
PRINT_BOTH_SIDES	BOTHSIDES	VARCHAR(7)	How the information prints. ***FORMDF** The file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, and *AFPDSLINE printer device type files. ***NO** The printing on the page is on one side only. ***YES** The printing is on both sides of the page with the top of each page the same for both sides. ***TUMBLE** The printing is on both sides with the top of one printed page at the opposite end from the top of the other printed page.
FILE_AVAILABLE	FILEAVAIL	VARCHAR(8)	The time when this file becomes available to an output device for processing. ***IMMED** The file is available as soon as the file is opened. ***FILEEND** The file is available as soon as the file is closed. ***JOBEND** The file is available when the job that owns the file is completed.
STARTING_PAGE	STARTPAGE	VARCHAR(10)	The page at which printing is to start for the file. Can contain the following special value: ***ENDPAGE** Printing starts with the last page.

Table 104. OUTPUT_QUEUE_ENTRIES view (continued)

Column Name	System Column Name	Data Type	Description
ENDING_PAGE	ENDPAGE	VARCHAR(10)	The page at which printing is to end for the file. Can contain the following special value: ***END** Printing ends with the last page.
DEVICE_FILE_LIBRARY	DEVLIB	VARCHAR(10)	The name of the library that contains the device file.
DEVICE_FILE_NAME	DEVFILE	VARCHAR(10)	The name of the device file used to create the spooled file.
PROGRAM_THAT_OPENED_ FILE_LIBRARY	LIBOPEN	VARCHAR(10) Nullable	The name of the library that contains the program that opened the file. Contains null when the program is not known.
PROGRAM_THAT_OPENED_ FILE_NAME	PGMOPEN	VARCHAR(10) Nullable	The name of the program that opened the spooled file. Contains null when the program is not known.
FORM_DEFINITION_LIBRARY	FORMLIB	VARCHAR(10) Nullable	The name of the library that contains the form definition. Contains null if FORM_DEFINITION_NAME is a special value or if no form definition is specified for this spooled file.
FORM_DEFINITION_NAME	FORMNAME	VARCHAR(10) Nullable	The name of the form definition to use for this print request. Can contain one of the following special values: ***DEVD** The form definition in the printer device description will be used. ***INLINE** The form definition defined in the spooled file data stream will be used. ***INLINED** The form definition defined in the spooled file data stream will be used. If a form definition is not found, the form definition in the printer device description will be used. **F1DFLT** The form definition defined in the spooled file data stream will be used. Contains null when no form definition is specified for this spooled file.
PAGE_DEFINITION_LIBRARY	PAGELIB	VARCHAR(10) Nullable	The name of the library containing the page definition. Contains the null value for *LINE or *AFPDSLINE printer device type files.
PAGE_DEFINITION_NAME	PAGENAME	VARCHAR(10) Nullable	The name of the page definition to use for the file. Contains the null value for *LINE or *AFPDSLINE printer device type files.
FRONT_OVERLAY_LIBRARY	FRONTLIB	VARCHAR(10) Nullable	The name of the library containing the front overlay. Can contain one of these special values: ***CURLIB** The current ibrary is searched the front overlay. ***LIBL** The library list is used to locate the front overlay. Contains null when FRONT_OVERLAY_NAME is *NONE.
FRONT_OVERLAY_NAME	FRONTNAME	VARCHAR(10)	The name of the front overlay. Can contain the following special value: ***NONE** The file does not use the front overlay.
BACK_OVERLAY_LIBRARY	BACKLIB	VARCHAR(10) Nullable	The name of the library containing the back overlay. Contains null when BACK_OVERLAY_NAME is a special value.
BACK_OVERLAY_NAME	BACKNAME	VARCHAR(10)	The name of the back overlay. Can contain the following special values: ***FRONTOVL** The back overlay is the same as the front overlay. ***NONE** The file does not use the back overlay.

Table 104. OUTPUT_QUEUE_ENTRIES view (continued)

Column Name	System Column Name	Data Type	Description
CHARACTER_SET_LIBRARY	CHRSETLIB	VARCHAR(10) Nullable	The name of the library containing the font character set object. Can contain one of these special values: ***CURLIB** The current library is searched for the font character set object. ***LIBL** The library list is used to locate the font character set object. Contains null when CHARACTER_SET_NAME is *FONT.
CHARACTER_SET_NAME	CHRSETNAME	VARCHAR(10)	The name of the font character set object used to print this file. Can contain the following special value: ***FONT** The information specified on the font parameter is used instead of the character set and code page.
CODE_PAGE_LIBRARY	CODELIB	VARCHAR(10) Nullable	The name of the library containing the code page used to print this spooled file. Can contain one of these special values: ***CURLIB** The current library is searched for the code page name. ***LIBL** The library list is used to locate the code page name. Contains null when no code page is specified for this spooled file.
CODE_PAGE_NAME	CODENAME	VARCHAR(10) Nullable	The name of the code page used to print this spooled file. Contains null when no code page is specified for this spooled file.
CHARACTER_SET_POINTSIZE	CHARSIZE	DECIMAL(5,1) Nullable	The point size in which this file's characters should be printed. Contains null if the character set does not have a point size.
CODED_FONT_LIBRARY	FONTLIB	VARCHAR(10) Nullable	The name of the library containing the coded font used to print this spooled file. Can contain one of these special values: ***CURLIB** The current library is searched for the coded font. ***LIBL** The library list is used to locate the coded font. Contains null when CODED_FONT_NAME is *FNTCHRSET.
CODED_FONT_NAME	FONTNAME	VARCHAR(10)	The name of the coded font used to print this spooled file. Can contain the following special value: ***FNTCHRSET** The values used are the values specified on the character set name and library name and code page name and library name fields.
CODED_FONT_POINTSIZE	FONTSIZE	DECIMAL(5,1) Nullable	The point size in which this file's characters should be printed. Contains null if the coded font does not have a point size.
MULTIBYTE_DATA	MULTIBYTE	VARCHAR(10)	Whether the file can contain double-byte character set (DBCS) data, Unicode data, or both. Values are *YES and *NO.
DBCS_CODED_FONT_LIBRARY	DBCSLIB	VARCHAR(10) Nullable	The name of the library containing the DBCS-coded font. Can contain one of these special values: ***CURLIB** The current library is searched for the DBCS-coded font. ***LIBL** The library list is used to locate the DBCS-coded font. Contains null when DBCS_CODED_FONT_NAME is *SYSVAL.
DBCS_CODED_FONT_NAME	DBCSNAME	VARCHAR(10)	The name of the DBCS-coded font used to print DBCS-coded data on printers configured as AFP(*YES). Can contain the following special value: ***SYSVAL** The DBCS-coded font specified in the system value is used.

Table 104. OUTPUT_QUEUE_ENTRIES view (continued)

Column Name	System Column Name	Data Type	Description
DBCS_CODED_FONT_POINTSIZE	DBCSSIZE	DECIMAL(5,1) Nullable	The point size in which this file's DCBS characters should be printed. Contains null if the DBCS-coded font does not have a point size.

Example

Find the 100 largest spool files in the QEZJOBLOG output queue.

```
SELECT * FROM QSYS2.OUTPUT_QUEUE_ENTRIES
  WHERE OUTPUT_QUEUE_NAME = 'QEZJOBLOG'
  ORDER BY SIZE DESC
  FETCH FIRST 100 ROWS ONLY
```

OUTPUT_QUEUE_ENTRIES_BASIC view

The OUTPUT_QUEUE_ENTRIES_BASIC view returns one row for each spooled file in every output queue. This view uses the QSYS2.OUTPUT_QUEUE_ENTRIES table function with `DETAILED_INFO => 'NO'`.

The schema is QSYS2.

Rows will be returned for spooled files when the caller has:

- Execute authority to the output queue library and
  - Read authority to the output queue object, or
  - *JOBCTL special authority and the output queue has OPRCTL(*YES), or
  - *SPLCTL special authority

To achieve the best performance when querying the OUTPUT_QUEUE_ENTRIES_BASIC view, the use of a WHERE clause is recommended if you are interested in examining specific output queue libraries or output queues. OUTPUT_QUEUE_ENTRIES_BASIC typically performs much better than OUTPUT_QUEUE_ENTRIES. OUTPUT_QUEUE_ENTRIES should only be used when OUTPUT_QUEUE_ENTRIES_BASIC does not include the columns needed by the query.

The following table describes the columns in the view. The system name is OUTQ_INFOB. The schema is QSYS2.

Table 105. OUTPUT_QUEUE_ENTRIES_BASIC view

Column Name	System Column Name	Data Type	Description
OUTPUT_QUEUE_NAME	OUTQ	VARCHAR(10)	Name of the output queue containing the spooled file.
OUTPUT_QUEUE_LIBRARY_NAME	OUTQLIB	VARCHAR(10)	The name of the library that contains the output queue.
CREATE_TIMESTAMP	CREATED	TIMESTAMP	The timestamp when the file was created.
SPOOLED_FILE_NAME	SPOOLNAME	VARCHAR(10)	The file name that was specified by the user program when the file was created, or the name of the device file used to create this file.
USER_NAME	USER_NAME	VARCHAR(10)	The name of the user profile that produced the file.
USER_DATA	USER_DATA	VARCHAR(10) Nullable	The user-specified data that describes this file. Contains null if there is no user-specified data.

Table 105. OUTPUT_QUEUE_ENTRIES_BASIC view (continued)

Column Name	System Column Name	Data Type	Description
STATUS	STATUS	VARCHAR(15)	Status of the spooled file. **CLOSED** The file has been completely processed by a program but SCHEDULE(*JOBEND) was specified and the job that produced the file has not yet finished. **DEFERRED** Printing of the file has been deferred. **DELETED** The file has been deleted. **HELD** The file has been held. **MESSAGE WAITING** This file has a message which needs a reply or an action to be taken. **OPEN** The file has not been completely processed and is not ready to be selected by a writer. **PENDING** The file is pending to be printed. **PRINTING** The file has been completely sent to the printer but print complete status has not been sent back. **READY** The file is available to be written. **SAVED** The file has been printed and then saved. This file remains saved until it is released. **SENDING** The file is being sent or has been sent to a remote system. **WRITING** This file is currently being produced by the writer.
SIZE	SIZE	INTEGER	The size of the spooled file, in kilobytes.
TOTAL_PAGES	PAGES	INTEGER	The total number of pages in the file.
COPIES	COPIES	SMALLINT	The number of copies remaining to print.
FORM_TYPE	FORM_TYPE	VARCHAR(10)	The type of form that should be loaded in the printer to print this file.
JOB_NAME	JOB_NAME	VARCHAR(28)	The qualified job name that produced the file.
DEVICE_TYPE	DEVTYPE	VARCHAR(10)	The type of data stream used to represent the file. ***AFPDS** Advanced Function Presentation data stream ***AFPDSLINE** AFPDS data mixed with 1403 line data ***IPDS** Intelligent printer data stream ***LINE** 1403 line data ***SCS** Systems Network Architecture (SNA) character stream ***USERASCII** ASCII data
OUTPUT_PRIORITY	OUTPTY	SMALLINT	The priority of the spooled file.
FILE_NUMBER	FILENUM	INTEGER	The spooled file number of the specified file.
SYSTEM	SYSTEM	VARCHAR(8)	The name of the system where the job that created the spooled file ran.

Examples

- Find the 100 largest spool files in the QEZJOBLOG output queue.

```
SELECT * FROM QSYS2.OUTPUT_QUEUE_ENTRIES_BASIC
  WHERE OUTPUT_QUEUE_NAME = 'QEZJOBLOG'
  ORDER BY SIZE DESC
  FETCH FIRST 100 ROWS ONLY
```

- Find the top 10 consumers of SPOOL storage.

```
SELECT USER_NAME, SUM(SIZE) AS TOTAL_SPOOL_SPACE
  FROM QSYS2.OUTPUT_QUEUE_ENTRIES_BASIC
  WHERE USER_NAME NOT LIKE 'Q%'
  GROUP BY USER_NAME
  ORDER BY TOTAL_SPOOL_SPACE DESC LIMIT 10;
```

OUTPUT_QUEUE_INFO view

The OUTPUT_QUEUE_INFO view returns one row for each output queue.

The values returned for the columns in the view are similar to the values returned by the Retrieve Output Queue Information (QSPROUTQ) API. Refer to the API for more detailed information.

Authorization: Rows will be returned for output queues when the caller has:

- Execute authority to the output queue library and
  - Read authority to the output queue object, or
  - *JOBCTL special authority and the output queue has OPRCTL(*YES), or
  - *SPLCTL special authority

The following table describes the columns in the view. The system name is OUTQ_DTL. The schema is QSYS2.

Table 106. OUTPUT_QUEUE_INFO view

Column Name	System Column Name	Data Type	Description
OUTPUT_QUEUE_NAME	OUTQ	VARCHAR(10)	Name of the output queue.
OUTPUT_QUEUE_LIBRARY_NAME	OUTQLIB	VARCHAR(10)	The name of the library that contains the output queue.
NUMBER_OF_FILES	FILES	INTEGER	The total number of spooled files currently on this output queue.
NUMBER_OF_WRITERS	WRITERS	INTEGER	The number of printer writers that have been started to this output queue.
WRITERS_TO_AUTOSTART	AUTOSTART	INTEGER	The number of remote printer writers to autostart to this output queue at system IPL.
PRINTER_DEVICE_NAME	DEV_NAME	VARCHAR(10) Nullable	The name of the printer device. If more than one writer is started, this is the printer device name of the first writer. Contains the null value if WRITER_TYPE is not PRINTER.
ORDER_OF_FILES	FILE_ORDER	VARCHAR(7)	The order of the spooled files on the output queue. ***FIFO** The queue is first-in first-out for each file. That is, on the queue, new spooled files are placed behind all other spooled files that have the same priority. ***JOBNBR** The queue entries for the spooled files are sorted in priority sequence using the job number (the date and time that the job entered the system) of the job that created the spooled file.

Table 106. OUTPUT_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
DISPLAY_ANY_FILE	ANYFILE	VARCHAR(6)	Whether users who have authority to read this output queue can display the output data of any output file on this queue, or only the data in their own files. ***NO** Users authorized to the queue can only display, copy, or send their own spooled files, unless one of the following applies: • they have *SPLCTL special authority, or • they have *JOBCTL special authority and OPERATOR_CONTROLLED is *YES. ***OWNER** Only the owner of a file or a user with *SPLCTL authority can display, copy, send, or move their own spooled files to another output queue. ***YES** Any user having authority to read the queue can display, copy, or send the data of any file on the queue.
JOB_SEPARATORS	JOB_SEP	VARCHAR(4)	The number of job separators (0-9) to be placed at the beginning of the output for each job having spooled file entries on this output queue. Can also contain the following special value: ***MSG** No job separators are used; instead a message is sent to the writer's message queue at the end of each job indicating that the output can be removed.
MAXIMUM_PAGES	MAX_PAGES	INTEGER Nullable	Only spooled files with this number of pages or less will print between MAXIMUM_PAGES_STARTING_TIME and MAXIMUM_PAGES_ENDING_TIME. If more than one set of maximum spooled file size values is defined for this output queue, only information for the first set is returned. Contains the null value if no maximum spooled file size is defined.
MAXIMUM_PAGES_STARTING_TIME	MAX_START	TIME Nullable	The starting time, in local job time, that spooled files exceeding MAXIMUM_PAGES will be restricted from printing. If a spooled file exceeds the page limit it will be in deferred status until ENDING_TIME. Contains the null value if no maximum spooled file size is defined.
MAXIMUM_PAGES_ENDING_TIME	MAX_END	TIME Nullable	The ending time, in local job time, when spooled files exceeding MAXIMUM_PAGES will be allowed to print. Contains the null value if no maximum spooled file size is defined.
OPERATOR_CONTROLLED	OPR_CTRL	VARCHAR(4)	Whether users with job control authority are allowed to manage or control the files on this queue. Users have job control authority if SPCAUT(*JOBCTL) is specified in their user profile. ***NO** This queue and its entries cannot be controlled or changed by users with job control authority unless they also have some other special authority. ***YES** Users with job control authority can control the queue and make changes to the files on the queue.
AUTHORITY_TO_CHECK	ALL_AUTH	VARCHAR(7)	Indicates what type of authorities to the output queue allow the user to control all the files on the queue. ***DTAAUT** Any user with *READ, *ADD, and *DELETE authority to the output queue can control all output files on the queue. ***OWNER** Only the owner of the output queue can control all the output files on the queue.
DATA_QUEUE_LIBRARY	DTAQ_LIB	VARCHAR(10) Nullable	The name of the library containing the data queue. Contains the null value if no data queue is associated with this output queue.
DATA_QUEUE_NAME	DTAQ_NAME	VARCHAR(10) Nullable	The name of the data queue associated with this output queue. Contains the null value if no data queue is associated with this output queue.

Table 106. OUTPUT_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
OUTPUT_QUEUE_STATUS	STATUS	VARCHAR(8)	The status of the output queue. **HELD** The queue is held. **RELEASED** The queue is released.
WRITER_JOB_NAME	WRITER_JOB	VARCHAR(28) Nullable	The qualified job name of the writer job. If more than one writer is started, this is the name of the first writer. Contains the null value if a writer job is not started for this queue.
WRITER_JOB_STATUS	WRITER_STS	VARCHAR(4) Nullable	The status of the writer job. If more than one writer is started, this is the status of the first writer. **END** The writer job has ended. **HLD** The writer job is held. **JOBQ** The writer job is on the job queue. **MSGW** The writer job is waiting for a message response. **STR** The writer job is started for the output queue. Contains the null value if a writer job is not started for this queue.
WRITER_TYPE	WRITER_TYP	VARCHAR(7) Nullable	The type of writer started for this output queue. **PRINTER** Printer writer. **REMOTE** Remote writer. Contains the null value if a writer job is not started for this queue.
SPOOLED_FILE_ASP_ATTRIBUTE	ASP_ATTR	VARCHAR(8)	The auxiliary storage pool (ASP) where the spooled files are to reside. ***OUTQASP** The spooled files reside in the auxiliary storage pool in which the output queue resides. ***SYSTEM** The spooled files reside in the system auxiliary storage pool.
SPOOLED_FILE_ASP_NUMBER	ASPNUM	INTEGER	The number of the auxiliary storage pool (ASP) where the spooled files reside.
SPOOLED_FILE_ASPGRP	ASPGRP	VARCHAR(10) Nullable	The name of the auxiliary storage pool (ASP) device where the spooled files reside. Can also contain the following special value: ***SYSBAS** The spooled files resides in the system ASP (ASP 1) or one of the defined basic user ASPs (ASPs 2-32). Contains the null value if the name is not available.
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	The text description of the output queue. Contains the null value if the output queue has no description.
MESSAGE_QUEUE_LIBRARY	MSGQ_LIB	VARCHAR(10) Nullable	The name of the library containing the message queue. Can contain the following special value: ***LIBL** The library list is searched to find the message queue. Contains the null value if the output queue is not a remote output queue or if WRITER_TYPE is PRINTER.
MESSAGE_QUEUE_NAME	MSGQ_NAME	VARCHAR(10) Nullable	The name of the message queue to which messages, created by the remote writer started to this output queue, are sent. Contains the null value if the output queue is not a remote output queue or if WRITER_TYPE is PRINTER.

Table 106. OUTPUT_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
HOST_PRINT_TRANSFORM	TRANSFORM	VARCHAR(4) Nullable	Whether to use the host print transform function to transform a spooled file. ***NO** Do not transform data streams using host print transform. ***YES** Transform data streams using host print transform. Contains the null value if NETWORK_CONNECTION_TYPE is *SNA and USER_DRIVER_PROGRAM_NAME is null.
IMAGE_CONFIGURATION_NAME	IMAGE_NAME	VARCHAR(10) Nullable	The name of the image configuration. Contains the null value if no image configuration is used when transforming the spooled file before sending.
MANUFACTURER_TYPE_AND_MODEL	TYPE_MODEL	VARCHAR(17) Nullable	The manufacturer, type, and model for a printer using the host print transform function. See Printer Model Settings for Host Print Transform (HPT) in the IBM Support Portal for the list of supported values. Contains the null value when NETWORK_CONNECTION_TYPE is *SNA, or when NETWORK_CONNECTION_TYPE is *IP and HOST_PRINT_TRANSFORM is *NO.
WORKSTATION_CUSTOMIZING_OBJECT_LIBRARY	CUSTOM_LIB	VARCHAR(10) Nullable	The library name for the workstation customizing object. Contains the null value if there is no workstation customizing object or if NETWORK_CONNECTION_TYPE is *SNA and USER_DRIVER_PROGRAM_NAME is null.
WORKSTATION_CUSTOMIZING_OBJECT_NAME	CUSTOM_NAM	VARCHAR(10) Nullable	The name of an object that consists of a table of attributes used to customize a given ASCII device. Contains the null value if there is no workstation customizing object or if NETWORK_CONNECTION_TYPE is *SNA and USER_DRIVER_PROGRAM_NAME is null.
NETWORK_CONNECTION_TYPE	NET_TYPE	VARCHAR(7) Nullable	The type of network connection to the remote system. ***IP** The TCP/IP network is used as the connectivity to the remote system. ***SNA** The SNADS network is used as the connectivity to the remote system. ***USRDFN** A user-defined connectivity is used as the connectivity to the remote system. Contains the null value if the output queue is not a remote output queue.
DESTINATION_TYPE	DEST_TYPE	VARCHAR(8) Nullable	The type of destination system that spooled files on this output queue are being sent to. ***NDS** The destination is Novell NetWare 3 or 4, and the connection type is *USRDFN. ***OS400** The destination system is an IBM i. ***OTHER** The destination system does not match any of the other special values. This is commonly used when the destination is a printer. ***PSF2** The destination system is a PC using Print Services Facility/2. ***S390** This destination system is a System/390® system. Contains the null value if the output queue is not a remote output queue.
REMOTE_SYSTEM_NAME	REMOTE_NAM	VARCHAR(255) Nullable	The name of the remote system. Contains the null value if the output queue is not a remote output queue.

Table 106. OUTPUT_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
REMOTE_PRINTER_QUEUE	REMOTE_PRT	VARCHAR(255) Nullable	The name of the remote printer. Can also contain one of these special values: ***SYSTEM** The default system printer on the remote system will determine the printer queue. ***USER** The user profile that creates the spooled file will determine the user ID on the remote system that it is sent to. Contains the null value if the output queue is not a remote output queue.
DESTINATION_OPTIONS	DEST_OPT	VARCHAR(128) Nullable	Destination-dependent options that are specific to a particular implementation of an LPR Print Server. Can also contain the special values: ***NOWAIT** The remote writer will not wait for confirmation that the destination system has finished processing the spooled file. ***USRDFNTXT** Use the value for the user-defined text of the user profile when the spooled file was created. Contains the null value if the output queue is not a remote output queue.
USER_DRIVER_PROGRAM_ LIBRARY	UDP_LIB	VARCHAR(10) Nullable	The name of the library that contains the user driver program. Can also be one of these special values: ***CURLIB** The current library for the job is used to locate the user driver program. ***LIBL** The library list used to locate the user driver program. Contains the null value if no user driver program is specified.
USER_DRIVER_PROGRAM_NAME	UDP_NAME	VARCHAR(10) Nullable	The name of the user-specified driver program that is used to process the spooled files on the output queue. Contains the null value if no user driver program is specified.
USER_DEFINED_OBJECT_LIBRARY	UDO_LIB	VARCHAR(10) Nullable	The name of the library that contains the user-defined object. Can also be one of these special values: ***CURLIB** The current library for the job is used to locate the user-defined object. ***LIBL** The library list used to locate the user-defined object. Contains the null value if no user-defined object is specified.
USER_DEFINED_OBJECT_NAME	UDO_NAME	VARCHAR(10) Nullable	The name of the user-defined object that is used by user applications or user-specified programs that process spooled files. Contains the null value if no user-defined object is specified.
USER_DEFINED_OBJECT_TYPE	UDO_TYPE	VARCHAR(7) Nullable	The type of the user-defined object. ***DTAARA** Data area. ***DTAQ** Data queue. ***FILE** File. ***PSFCFG** PSF configuration object. ***USRIDX** User index. ***USRQ** User queue. ***USRSPC** User space. Contains the null value if no user-defined object is specified.

Table 106. OUTPUT_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
DATA_TRANSFORM_PROGRAM_ LIBRARY	DTP_LIB	VARCHAR(10) Nullable	The name of the library that contains the data transform program. Can also be one of these special values: ***CURLIB** The current library for the job is used to locate the data transform program. ***LIBL** The library list used to locate the data transform program. Contains the null value if no data transform program is specified.
DATA_TRANSFORM_PROGRAM_ NAME	DTP_NAME	VARCHAR(10) Nullable	The name of the user-specified data transform program that is used by the driver program. Contains the null value if no data transform program is specified.
USER_DEFINED_OPTION_1	UDEF_OPT1	VARCHAR(10) Nullable	The first user-defined option. Contains the null value if there are no user-defined options.
USER_DEFINED_OPTION_2	UDEF_OPT2	VARCHAR(10) Nullable	The second user-defined option. Contains the null value if there are not at least two user-defined options.
USER_DEFINED_OPTION_3	UDEF_OPT3	VARCHAR(10) Nullable	The third user-defined option. Contains the null value if there are not at least three user-defined options.
USER_DEFINED_OPTION_4	UDEF_OPT4	VARCHAR(10) Nullable	The fourth user-defined option. Contains the null value if there are not at least four user-defined options.
USER_DEFINED_DATA	UDEF_DATA	VARBINARY(5000) Nullable	Data defined by the user to be used by user applications or user-specified programs that process spooled files. Contains the null value if there is no user-defined data.
LDAP_PUBLISHING_STATUS	PUBLISHED	VARCHAR(3)	Whether the output queue is published in the network directory. **NO** Output queue is not published. **YES** Output queue is published.
FORMS_CONTROL_BUFFER	FORMS_BUF	VARCHAR(8) Nullable	The forms control buffer (FCB) for files sent to a VM/MVS host system. Contains either the name of the FCB or one of the following special values: ***PRTF** The first 8 characters of the printer file used to spool the file determines the name of the FCB. ***USRDTA** The first 8 characters of the user data (USRDATA) spooled file attribute determines the name of the FCB. If the user data is blank, no FCB is used. Contains the null value if no FCB is used when sending spooled files or if NETWORK_CONNECTION_TYPE is not *SNA or DESTINATION_TYPE is not *S390.
VM_MVS_CLASS	VM_CLASS	CHAR(1) Nullable	The VM/MVS SYSOUT class for distributions sent to a VM host system or to a MVS host system. Values are A-Z, 0-9 to indicate the distribution class. Contains the null value if not defined for this output queue.

Example

```
SELECT * FROM QSYS2.OUTPUT_QUEUE_INFO
```

Storage Services

These views provide information about storage and storage devices.

ASP_INFO view

The ASP_INFO view returns information about auxiliary storage pools (ASPs).

The values returned for the columns in the view are similar to the values returned by the Work with Configuration Status (WRKCFGSTS) CL command and the Open List of ASPs (QYASPOL) API.

Authorization: None required.

The following table describes the columns in the view. The system name is ASP_INFO. The schema is QSYS2.

Table 107. ASP_INFO view

Column Name	System Column Name	Data Type	Description
DEVICE_DESCRIPTION_NAME	DEVD_NAME	VARCHAR(10) Nullable	The name of the device description that brought the independent ASP (IASP) to varyon/active state. Contains the null value if the ASP is not an IASP or if it is an IASP and ASP_STATE is `VARIED OFF`.
ASP_NUMBER	ASP_NUM	INTEGER	A unique identifier for an ASP. Possible values are 1 through 255. **1** The system ASP **2-32** User ASPs **33-255** IASPs
ASP_STATE	ASP_STATE	VARCHAR(10)	The device configuration status of an ASP. **ACTIVE** The status of the ASP is active. **AVAILABLE** The status of the ASP is available. **NONE** There is no status. This value is used for the system ASP and any basic user ASPs. **VARIED OFF** The status of the ASP is varyoff. **VARIED ON** The status of the ASP is varyon.
ASP_TYPE	ASP_TYPE	VARCHAR(9) Nullable	The use that is assigned to the ASP. **PRIMARY** The ASP is a primary ASP. **SECONDARY** The ASP is a secondary ASP. **SYSTEM** The ASP is the system ASP. **UDFS** The ASP is a user-defined file system ASP. **USER** The ASP is a user ASP. Contains the null value for an IASP when the type cannot be determined.
RDB_NAME	RDB_NAME	VARCHAR(18) Nullable	The name that is assigned to the database that this ASP defines. Contains the null value if ASP_TYPE is not `PRIMARY` or `SECONDARY`.
NUMBER_OF_DISK_UNITS	DISK_UNITS	INTEGER	The total number of disk units in the ASP. If mirroring is active for disk units within the ASP, the mirrored pair of units is counted as one.

Table 107. ASP_INFO view (continued)

Column Name	System Column Name	Data Type	Description
DISK_UNITS_PRESENT	PRESENT	VARCHAR(4)	Indicates whether disk units in the ASP were found. **ALL** All disk units were found. **NONE** No disk units were found. **SOME** The disk unit that is used to provide the identity of the ASP was found but some other disk units were not found.
TOTAL_CAPACITY	TOTCAP	BIGINT Nullable	The total number of used and unused megabytes in the ASP. A special value of -2 is returned if the size of this field is exceeded. Contains the null value if the capacity cannot be determined.
TOTAL_CAPACITY_AVAILABLE	TOTCAPA	BIGINT Nullable	The total number of unused megabytes in the ASP. A special value of -2 is returned if the value was too big to return. Contains the null value if the capacity cannot be determined.
PROTECTED_CAPACITY	PROTCAP	BIGINT Nullable	The total number of used and unused megabytes in the ASP that are protected by mirroring or device parity. A special value of -2 is returned if the value was too big to return. Contains the null value if the capacity cannot be determined.
PROTECTED_CAPACITY_AVAILABLE	PROTCAPA	BIGINT Nullable	The number of unused megabytes in the ASP that are protected by mirroring or device parity. A special value of -2 is returned if the value was too big to return. Contains the null value if the capacity cannot be determined.
UNPROTECTED_CAPACITY	UNPROTCAP	BIGINT Nullable	The total number of used and unused megabytes in the ASP that are not protected by mirroring or device parity. A special value of -2 is returned if the value was too big to return. Contains the null value if the capacity cannot be determined.
UNPROTECTED_CAPACITY_ AVAILABLE	UNPROTCAPA	BIGINT Nullable	The number of unused megabytes in the ASP that are not protected by mirroring or device parity. A special value of -2 is returned if the value was too big to return. Contains the null value if the capacity cannot be determined.
SYSTEM_STORAGE	SYS_STG	INTEGER Nullable	The amount of storage in megabytes currently allocated in the ASP for operating system use. Contains the null value if this is not the system ASP.
OVERFLOW_STORAGE	OVER_STG	BIGINT Nullable	The number of megabytes of storage that has overflowed from the user ASP into the system ASP. A special value of -2 is returned if the value was too big to return. Contains the null value if this is an IASP.
STORAGE_THRESHOLD_PERCENTAGE	THRESHOLD	INTEGER	When the storage in the ASP reaches this percentage, a warning message is sent to the QSYSOPR message queue. When this percentage is reached for the system ASP (ASP 1), message CPF0907 is sent. When this percentage is reached for a user ASP, message CPI0953 is sent.

Table 107. ASP_INFO view (continued)

Column Name	System Column Name	Data Type	Description
OVERFLOW_RECOVERY_RESULT	OVER_RES	VARCHAR(7) Nullable	An indicator of the result of the ASP overflow recovery operation, which is performed during IPL at the user's request. When this operation is requested, an attempt is made to recover the user ASP from an overflow condition by moving overflowed auxiliary storage from the system ASP back to the user ASP during the storage management recovery step of an IPL. **CANCEL** ASP overflow recovery was canceled prior to completion. **FAIL** ASP overflow recovery failed due to insufficient space in the user ASP. **SUCCESS** All overflowed storage was successfully moved. Contains the null value if this is an IASP.
ERROR_LOG_SPACE	ERR_SPACE	INTEGER Nullable	The number of megabytes of auxiliary storage allocated to the error log. Contains the null value if this is not the system ASP.
MACHINE_LOG_SPACE	LOG_SPACE	INTEGER Nullable	The number of megabytes of auxiliary storage allocated to the machine log. Contains the null value if this is not the system ASP.
MACHINE_TRACE_SPACE	TRC_SPACE	INTEGER Nullable	The number of megabytes of auxiliary storage allocated to the machine trace. Contains the null value if this is not the system ASP.
MAIN_STORAGE_DUMP_SPACE	MSD_SPACE	INTEGER Nullable	The number of megabytes of auxiliary storage allocated to the main storage dump space. Contains the null value if this is not the system ASP.
MICROCODE_SPACE	MIC_SPACE	INTEGER Nullable	The number of megabytes of auxiliary storage allocated to the microcode and space used by the microcode. Contains the null value if this is an IASP that is varied off.
END_IMMEDIATE	END_IMMED	VARCHAR(3) Nullable	This column only applies to the system ASP (ASP 1). **NO** If a request for space in the system ASP cannot be satisfied because there is not enough storage, the system will be allowed to continue running. **YES** If a request for space in the system ASP cannot be satisfied because there is not enough storage, the system will be ended immediately. Contains the null value if this is not the system ASP.

Table 107. ASP_INFO view (continued)

Column Name	System Column Name	Data Type	Description
COMPRESSION_RECOVERY_POLICY	COMP_RECOV	VARCHAR(18)	The compression recovery policy for the ASP. If the ASP has compressed drives as part of its configuration, this value controls how overflow situations are handled for this ASP. The following policies allow the user to control what is done when the ASP appears full. **OVERFLOW DELAY** When the ASP capacity is about to be exceeded, the operating system posts system reference code (SRC) A6xx 0277 in the system control panel and waits for a limited time for space to become available. If space becomes available before the limited time ends, the SRC is removed from the system control panel and normal operations resume. If space does not become available before the limited time ends, data overflows into the system ASP. **OVERFLOW IMMEDIATE** When the ASP capacity is about to be exceeded, the data immediately overflows into the system ASP. **WAIT** When the ASP capacity is about to be exceeded, the operating system posts SRC A6xx 0277 in the system control panel and waits indefinitely for space to become available. The user must take action before normal operation resumes. Possible actions include deleting objects from the ASP or changing the compression recovery policy to a value that allows the ASP to overflow.
COMPRESSED_DISK_UNITS	COMPRESSED	VARCHAR(4)	Whether there are compressed disk units in the ASP. **ALL** All disk units in this ASP are compressed. **NONE** No compressed disk units in this ASP. **SOME** Compressed and uncompressed disk units in this ASP.
CHANGES_WRITTEN_TO_DISK	WRITTEN	VARCHAR(3)	An indicator of whether all changes made the previous time the IASP was online were written to disk. Varyoff processing attempts to write changed IASP storage but, in some failures, it may not be successful. **NO** Not all changes were written to disk. **YES** All changes were written to disk.
MULTIPLE_CONNECTION_DISK_UNITS	MULT_CONN	VARCHAR(3)	A disk unit may have multiple resource names. Each resource name represents a unique connection to the disk unit. All active connections are used to communicate with the disk unit. This attribute indicates whether the disk unit has more than one connection. **NO** The disk unit has only one connection. **YES** The disk unit has more than one connection.

Table 107. ASP_INFO view (continued)

Column Name	System Column Name	Data Type	Description
BALANCE_STATUS	BALANCE	VARCHAR(8) Nullable	The current status of the balance function for this ASP. **COMPLETE** The ASP balance function has completed running. The ASP is completely balanced. **ENDED** The ASP balance function has run, but was ended before the ASP was completely balanced. The Start ASP Balance (STRASPBAL) command can be used to restart the balance function. **ENDING** The ASP balance function is currently in the process of ending. Either the time limit has run out or the End ASP Balance (ENDASPBAL) command was issued for this ASP. **NONE** No balance activity has occurred for this ASP. **RUNNING** The ASP balance function is currently running for this ASP. Contains the null value if `ASP_STATE` is not `ACTIVE` or `AVAILABLE` for IASP.

Table 107. ASP_INFO view (continued)

Column Name	System Column Name	Data Type	Description
BALANCE_TYPE	BAL_TYPE	VARCHAR(21) Nullable	The type of balance activity that is currently running or was done last. **CAPACITY BALANCING** Capacity balancing. Capacity balancing redistributes data so that the percentage of disk space used is the same on all disk units within the ASP. **CLEAR COLLECTION DATA** Clear collection data. Clear collection data removes the trace data created by running the Trace ASP Balance (TRCASPBAL) command. **HSM BALANCING** Hierarchical Storage Management (HSM) balancing. HSM balancing can be run only on an ASP that contains a mixture of high-performance and low-performance disk units. An example of low-performance disk units is compressed disk units. The HSM balance function moves high-use data to high-performance units and moves low-use data to low-performance units. The high-use and low-use data is identified by running the Trace ASP Balance (TRCASPBAL) command. **MOVE DATA** Move data. Move data is used to reduce the down time associated with removing a disk unit. The Check ASP Balance (CHKASPBAL) command can be used to determine which units are currently marked to no longer receive new allocations and to have their existing allocations moved to other disk units. **MP BALANCING** Media Preference (MP) balancing. MP balancing can be run only on an ASP that contains a mixture of Solid State Disk (SSD) units and Hard Disk Drive (HDD) units. The goal of the MP balance function is to have high-use data on SSD units and low-use data on HDD units. The high-use and low-use data is identified by running the Trace ASP Balance (TRCASPBAL) command. **NONE** No ASP balance activity was requested for the ASP. **USAGE BALANCING** Usage balancing. Usage balancing redistributes data so that the percentage of disk activity is the same on all disk units within the ASP. High-use and low-use data is identified by running the Trace ASP Balance (TRCASPBAL) command. Usage balancing moves data among the disk units, guided by the trace results, in an attempt to equalize the utilizations. Contains the null value if `ASP_STATE` is not `ACTIVE` or `AVAILABLE` for IASP.
BALANCE_DATA_MOVED	BAL_MOVED	BIGINT Nullable	The number of megabytes that have been moved by the balance function. A special value of -2 is returned if the value was too big to return. Contains the null value if `BALANCE_STATUS` is not `RUNNING`.
BALANCE_DATA_REMAINING	BAL_REMAIN	BIGINT Nullable	The number of megabytes that remain to be moved by the balance function before the move is considered complete. A special value of -2 is returned if the value was too big to return. Contains the null value if `BALANCE_STATUS` is not `RUNNING`.

Table 107. ASP_INFO view (continued)

Column Name	System Column Name	Data Type	Description
BALANCE_TIMESTAMP	BAL_TIME	TIMESTAMP(0) Nullable	The timestamp of the last status change for the balance function. Contains the null value when BALANCE_TYPE is NONE or the null value.
TRACE_STATUS	TRC_STATUS	VARCHAR(10) Nullable	The current status of the trace function. The trace gathers statistics about the data on the disk units within the ASP. This data is used by the balance functions. **CLEARING** The trace data for this ASP is being cleared. **COMPLETE 1** The trace function has completed running. The statistics for the ASP have been gathered and are ready for the balance function to start. **COMPLETE 2** The trace function has completed and the statistics for the ASP have been gathered. The ASP is ready for further collection or for the balance function to start. **ENDING** The trace function is currently in the process of ending. Either the time limit has run out or the trace was stopped through use of the Trace ASP Balance (TRCASPBAL) command. **NONE** There is no current trace data for this ASP. **RUNNING** The trace function is currently running for this ASP. Contains the null value if ASP_STATE is not ACTIVE or AVAILABLE for IASP.
TRACE_DURATION	TRC_DUR	INTEGER Nullable	The number of minutes that the trace function has run collecting data for this ASP. The trace can be run multiple times for an ASP. Contains the null value when TRACE_STATUS is NONE or the null value.
TRACE_TIMESTAMP	TRC_TIME	TIMESTAMP(0) Nullable	The timestamp of the last status change for the trace function. Contains the null value when TRACE_STATUS is NONE or the null value.
RESOURCE_NAME	RESOURCE	VARCHAR(10) Nullable	The resource name that identifies the ASP by which a collection of disks is known. Contains the null value for the system ASP, any user ASPs, and for an IASP where the name cannot be determined.
PRIMARY_ASP_RESOURCE_NAME	PRIMARY	VARCHAR(10) Nullable	The resource name of the primary ASP for a secondary ASP. Contains the null value if ASP_TYPE is not SECONDARY.

Example

- Show ASP information for the partition.

```
SELECT * FROM QSYS2.ASP_INFO;
```

ASP_VARY_INFO view

The ASP_VARY_INFO view returns one row for each step associated with a vary on or vary off operation for all independent ASP devices.

The values returned for the columns in the view are similar to the values returned by the Display ASP Status (DSPASPSTS) CL command.

Authorization: The privileges held by the authorization ID of the statement must have *USE authority to the independent ASP device description. If the user does not have *USE authority to all independent ASP device descriptions, a warning is returned to indicate that partial data is returned.

The following table describes the columns in the view. The system name is VARY_INFO. The schema is QSYS2.

Table 108. ASP_VARY_INFO view

Column Name	System Column Name	Data Type	Description
IASP_NAME	IASP_NAME	VARCHAR(10)	The name of the ASP device description.
OPERATION_NUMBER	OP_NUMBER	INTEGER	A value for an instance of a vary on or vary off operation, where the highest number is the most recent operation. The most recent 64 operations are returned.
OPERATION_TYPE	OP_TYPE	VARCHAR(8)	The type of vary operation. **VARY OFF** **VARY ON**
OPERATION_STATE	OP_STATE	VARCHAR(8)	The state of the entire operation. **ACTIVE** The operation is active. STEP_STATE shows the status of the steps that are part of the operation. **COMPLETE** The operation completed successfully. **FAILED** The operation failed to complete successfully.
STEP	STEP	VARGRAPHIC(50) CCSID 1200	The description of the operation step.
STEP_STATE	STEP_STATE	VARCHAR(8)	The state of the operation step. **ACTIVE** The step is active. **COMPLETE** The step completed successfully. **FAILED** The step failed to complete successfully.
START_TIMESTAMP	START	TIMESTAMP	The timestamp for the start of this operation step.
END_TIMESTAMP	END	TIMESTAMP Nullable	The timestamp for the end of this operation step. Contains the null value if the operation step has not completed or may never complete.
DURATION	DURATION	DECIMAL(12,6) Nullable	The time duration, in seconds, of this operation step. Contains the null value if the operation step has not completed or may never complete.
JOB_NAME	JOB_NAME	VARCHAR(28) Nullable	The qualified job name that initiated this vary operation. Contains the null value if the job name is not available.
IASP_NUMBER	IASPNUM	INTEGER	The number associated with the ASP device.

Example

- Return the steps from available vary on operations, listed from most expensive to least expensive.

```
SELECT * FROM QSYS2.ASP_VARY_INFO
  WHERE OPERATION_TYPE = 'VARY ON'
  ORDER BY IASP_NAME, DURATION DESC;
```

- Create a table to retain vary on historical data. Populate it with the current available values.

```
CREATE TABLE VARY_HISTORY AS
      (SELECT * FROM QSYS2.ASP_VARY_INFO) WITH DATA;
```

- Update the table that contains vary on historical data with any new rows.

```
MERGE INTO VARY_HISTORY H
      USING QSYS2.ASP_VARY_INFO N
      ON H.OPERATION_NUMBER = N.OPERATION_NUMBER
    WHEN NOT MATCHED THEN
      INSERT VALUES (N.IASP_NAME, N.OPERATION_NUMBER, N.OPERATION_TYPE,
                     N.OPERATION_STATE, N.STEP, N.STEP_STATE,
                     N.START_TIMESTAMP, N.END_TIMESTAMP, N.DURATION,
                     N.JOB_NAME, N.IASP_NUMBER);
```

MEDIA_LIBRARY_INFO view

The MEDIA_LIBRARY_INFO view returns information that can also be seen through the Work with Media Library Status (WRKMLBSTS) command interface.

The following table describes the columns in the view. The schema is QSYS2.

Table 109. MEDIA_LIBRARY_INFO view

Column Name	System Column Name	Data Type	Description
DEVICE_NAME	DEVICE	VARCHAR(10)	The name of the device.
DEVICE_STATUS	DEVICE_STS	VARCHAR(20)	The status of the device. The most common values are: **VARIED ON** The media library device is varied on. **VARIED OFF** The media library device is varied off. **ACTIVE** The resource is currently in use by a job under this media library. See List Configuration Descriptions API for a complete list of status values.
DEVICE_TYPE	DEVICE_TYP	VARCHAR(10)	The type of device. Contains the special value *RSRCNAME if the device type is determined by the resource in the RESOURCE_NAME column.
DEVICE_MODEL	DEVICE_MDL	VARCHAR(10)	The model number of the device. Contains the special value *RSRCNAME if the device model is determined by the resource in the RESOURCE_NAME column.
RESOURCE_NAME	RESOURCE	VARCHAR(10) Nullable	The name of the resource. Contains the null value if the DEVICE_STATUS column has a value of VARIED_OFF, or if the tape library does not have any associated tape resources.
RESOURCE_STATUS	RSRC_STS	VARCHAR(11) Nullable	The status of the resource. **OPERATIONAL** The resource is working and the system can address the tape drive resource. **ACTIVE** The resource is currently in use by a job under this media library. **UNAVAILABLE** The resource is currently not available because it may be in use by another object, another client, or DST. **FAILED** The resource is not operational and the system can no longer communicate with that resource. A hardware problem may have occurred. Contains the null value if the DEVICE_STATUS column has a value of VARIED_OFF, or if the tape library does not have any associated tape resources.

Table 109. MEDIA_LIBRARY_INFO view (continued)

Column Name	System Column Name	Data Type	Description
RESOURCE_ALLOCATION_STATUS	ALLOCATION	VARCHAR(11) Nullable	Current allocation status for the resource. **ALLOCATED** For a tape media library device the resource is exclusively assigned to this system and cannot be accessed by another system. For an optical media library device the drive is available for use by this media library. **UNPROTECTED** A tape resource is not exclusively assigned to this system. This resource can be assigned to this system when no other system has already assigned the resource. **DEALLOCATED** For a tape media library the resource is not assigned to this system and is not available to respond to requests. For an optical media library the device is not available for use by this media library. **STAND-ALONE** A tape resource is not available. The tape resource is reserved by a varied on stand-alone tape device description for non-library mode use. ***UNKNOWN** An optical media library is varied off or failed. The current allocation for a resource cannot be determined. Contains the null value if the DEVICE_STATUS column has a value of `VARIED_OFF`, or if the tape library does not have any associated tape resources.
RESOURCE_ALLOCATION _PRIORITY	ALLOC_PRTY	VARCHAR(4)	The priority of a job when requesting a resource. 1 is highest priority, 99 is lowest. Can contain the following special value: ***JOB** The priority of the job is used as the resource allocation priority.
INITIAL_MOUNT_WAIT_TIME	INIT_WAIT	VARCHAR(6)	The maximum amount of time a request will wait for allocation of a tape resource for the initial mount. Contains either a numeric string representing the number of minutes or one of the following special values: ***JOB** The allocation wait time is determined by the default wait time attribute of the job requesting the allocation, rounded up to the nearest minute. ***IMMED** The request will not wait for a tape resource to become available. ***NOMAX** The request will wait until a tape resource is available.
END_OF_VOLUME_MOUNT_WAIT _TIME	END_WAIT	VARCHAR(6)	The maximum amount of time a request will wait for allocation of a tape resource for the end of volume mount. Contains either a numeric string representing the number of minutes or one of the following special values: ***JOB** The allocation wait time is determined by the default wait time attribute of the job requesting the allocation, rounded up to the nearest minute. ***IMMED** The request will not wait for a tape resource to become available. ***NOMAX** The request will wait until a tape resource is available.
DEVICE_DESCRIPTION	DEVICE_DES	VARCHAR(50)	The text description of the device.

Example

Return information about all media library devices.

```
SELECT * FROM QSYS2.MEDIA_LIBRARY_INFO
```

SYSDISKSTAT view

The SYSDISKSTAT view contains information about disks.

The following table describes the columns in the view. The schema is QSYS2.

Table 110. SYSDISKSTAT view

Column Name	System Column Name	Data Type	Description
ASP_NUMBER	ASP_NUMBER	SMALLINT	Specifies the independent auxiliary storage pool (IASP) number.
DISK_TYPE	DISK_TYPE	VARCHAR(4)	Disk type number of the disk.
DISK_MODEL	DISK_MODEL	VARCHAR(4)	Model number of the disk.
UNIT_NUMBER	UNITNBR	SMALLINT	Unit number of the disk.
UNIT_TYPE	UNIT_TYPE	SMALLINT	Indicates the type of disk unit: **0** Not solid state disk **1** Solid state disk (SSD)
UNIT_STORAGE_CAPACITY	UNITSCAP	BIGINT	Unit storage capacity has the same value as the unit media capacity for configured disk units. This value is 0 for non-configured units.
UNIT_SPACE_AVAILABLE	UNITSPACE	BIGINT	Space (in bytes) available on the unit for use.
PERCENT_USED	PERCENTUSE	DECIMAL(7,3) Nullable	The percentage that the disk unit has been consumed.
UNIT_MEDIA_CAPACITY	UNITMCAP	BIGINT	Storage capacity (in bytes) of the unit.
LOGICAL_MIRRORED_PAIR_STATUS	MIRRORPS	CHAR(1) Nullable	Indicates the status of a mirrored pair of disks: **0** Indicates that one mirrored unit of a mirrored pair is not active. **1** Indicates that both mirrored units of a mirrored pair are active. Contains null if the unit is not mirrored.
MIRRORED_UNIT_STATUS	MIRRORUS	CHAR(1) Nullable	Indicates the status of a mirrored unit: **1** Indicates that this mirrored unit of a mirrored pair is active (online with current data). **2** Indicates that this mirrored unit is being synchronized. **3** Indicates that this mirrored unit is suspended. Contains null if the unit is not mirrored.

Example

- Return information about all disks.

```
SELECT * FROM QSYS2.SYSDISKSTAT
```

- Return information for all SSD units.

```
SELECT * FROM QSYS2.SYSDISKSTAT WHERE UNIT_TYPE = 1
```

SYSTMPSTG view

The SYSTMPSTG view contains one row for every temporary storage bucket that is tracking some amount of temporary storage across the system.

Temporary storage is application working storage that does not persist across a restart of the operating system. Accounting for all the temporary storage being used on the system is implemented using the concept of temporary storage buckets.

There are two types of temporary storage buckets:

- global buckets that are used to track temporary storage that is scoped to all jobs on the system.
- job buckets that are used to track temporary storage that is scoped to a single job.

Each bucket has a bucket number. Global buckets managed by the licensed internal code have bucket numbers from 1 to 4095. Global buckets managed by IBM i Work Management have bucket numbers from 4096 to 65535. Job buckets have numbers greater than 65535.

A job temporary storage bucket is assigned when the job starts and does not change for the life of the job. A job temporary storage bucket will normally be empty after the associated job ends and all working storage for the job is deleted or freed. If the job temporary storage bucket is empty after the job ends, the bucket becomes available to be associated with a new job. If the job associated with the job buckets ends and some temporary objects tracked to that job are not deleted, the job bucket will show a status of *ENDED as well as the date and time that the job ended. These job buckets identify jobs that are not deleting all of their temporary storage when the job ends.

Statistics for each job bucket indicate the current amount of storage (in bytes) used for temporary storage tracked by the bucket, the storage limit (in bytes) for disk storage used for temporary storage tracked by the bucket, and the peak amount of disk storage (in bytes) used for temporary storage tracked by the bucket. A job bucket does not include any temporary storage used for SQL query execution. For job buckets, the storage limit will reflect the MAXTMPSTG value of the class (*CLS) object specified when the job was submitted; a null value is returned if the job has a MAXTMPSTG value of *NOMAX.

The following table describes the columns in the view. The schema is QSYS2.

Table 111. SYSTMPSTG view

Column Name	System Column Name	Data Type	Description
BUCKET_NUMBER	BKTNBR	INTEGER	Number that uniquely identifies the temporary storage bucket.
GLOBAL_BUCKET_NAME	GLBBKTNAME	VARCHAR(30) Nullable	For global buckets, the name of the bucket. For job buckets, contains the null value.
JOBNAME	JOBNAME	VARCHAR(10) Nullable	For job buckets, the job name. For global buckets, contains the null value.
JOB_USER_NAME	JOBUSRNAME	VARCHAR(10) Nullable	For job buckets, the user profile under which the job is run. For global buckets, contains the null value.
JOB_NUMBER	JOBNBR	CHAR(6) Nullable	For job buckets, the job number assigned by the system. For global buckets, contains the null value.
BUCKET_CURRENT_SIZE	BKTCURSIZ	DECIMAL(23,0)	The current number of bytes of disk storage for this temporary storage bucket.
BUCKET_LIMIT_SIZE	BKTLMTSIZ	DECIMAL(23,0) Nullable	The current limit, in bytes, for the amount of disk storage for this temporary storage bucket. If the temporary storage bucket has no limit, contains the null value.
BUCKET_PEAK_SIZE	BKTPEAKSIZ	DECIMAL(23,0)	The largest number of bytes of disk storage for this temporary storage bucket. For global buckets, this is the peak amount of disk storage since the last restart of the operating system. For job buckets, this is the peak amount of disk storage since the job was started.

Table 111. SYSTMPSTG view (continued)

Column Name	System Column Name	Data Type	Description
JOB_STATUS	JOBSTS	VARCHAR(7) Nullable	For job buckets, indicates whether the bucket is associated with an active job or a job that ended without deleting all temporary objects associated with the job. ***ENDED** The job associated with this job bucket has ended. ***ACTIVE** The job associated with this job bucket is still active. For global buckets, contains the null value.
JOB_ENDED_TIME	JOBENDTIM	TIMESTAMP Nullable	For job buckets associated with jobs that have ended, indicates the timestamp of when the associated job ended. Contains the null value for global buckets and job buckets associated with active jobs.

USER_STORAGE view

The USER_STORAGE view contains details about storage by user profile.

The user storage consumption detail is determined by using Retrieve User Information (QSYRUSRI) API.

You must have *OBJOPR and *READ authority to a *USRPRF or it will not be returned. To see information for independent ASPs (iASPs), the iASP must be varied on.

User storage is broken down by SYSBAS and iASPs.

The following table describes the columns in the view. The schema is QSYS2.

Table 112. USER_STORAGE view

Column Name	System Column Name	Data Type	Description
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10) Nullable	User profile name.
ASPGRP	ASPGRP	VARCHAR(10) Nullable	Name of the independent ASP or *SYSBAS.
MAXIMUM_STORAGE_ALLOWED	MAXSTG	BIGINT Nullable	The maximum amount of auxiliary storage (in kilobytes) that can be assigned to store permanent objects owned by the user. Contains null if the user does not have a maximum amount of allowed storage.
STORAGE_USED	STGUSED	BIGINT Nullable	The amount of auxiliary storage (in kilobytes) occupied by the user's owned objects for this ASPGRP.

Example

Determine how much storage user SCOTTF has consumed.

```
SELECT * FROM QSYS2/USER_STORAGE
  WHERE USER_NAME = ‘SCOTTF’
```

System Health Services

For the most important system resources, the IBM i operating system automatically tracks the highest consumption and consumers.

The IBM i operating system is comprised of many products and components. As an integrated operating system, not only do the products and components frequently rely upon each other, but common building blocks and resources are used. Some of the resources are deemed to be critical because their proper use and consumption is directly related to achieving continued, normal operational behavior. The repository for this tracking lies within DB2 for i.

A table, a view, and global variables combine to provide information about limits on your system. Information about the important limits is logged in a DB2 for i supplied table named QSYS2/SYSLIMTBL. The QSYS2/SYSLIMITS view uses SYSLIMTBL and other DB2 resources to provide extended and formatted detail about these limits. You should generally work with the view rather than the underlying table. You can use DB2 for i provided global variables to control the number of rows kept for each type of limit in SYSLIMTBL.

The limits that are tracked are:

- ASP limits
  - Maximum number of spool files
- Database limits
  - Maximum number of all rows in a partition
  - Maximum number of valid rows in a partition
  - Maximum number of deleted rows in a partition
  - Maximum size of a table
  - Maximum number of overflow rows in a partition
  - Maximum number of variable-length segments
  - Maximum number of indexes over a partition
  - Maximum size of a *MAX4GB index
  - Maximum size of a *MAX1TB index
  - Maximum size of an encoded vector index
  - Maximum size of an extended dynamic package
- File system limits
  - Maximum number of object description entries in a library
  - Number of objects linked in a directory
  - Maximum number of directories linked in a directory
  - Maximum number of file system objects in *SYSBAS ASPs
  - Maximum number of file system objects in an independent ASP
  - Maximum number of document library objects in a folder
  - Number of document library objects in the system ASP
  - Maximum number of document library objects in a user ASP
  - Maximum number of bytes in a stream file
  - Maximum number of bytes in a document
- Job limits
  - Maximum number of rows locked in a unit of work
  - Maximum number of row change operations in a unit of work
- Journal limits
  - Maximum size of a journal receiver
  - Maximum number of objects that can be associated with a *MAX10M journal
  - Maximum number of objects that can be associated with a *MAX250K journal
  - Maximum sequence number of a *MAXOPT3 journal
  - Maximum sequence number of a *MAXOPT1 or *MAXOPT2 journal
- Object limits
  - Maximum number of members in a source physical file
- System limits
  - Maximum number of jobs

System limit alerts

Some system limits are instrumented by the IBM i operating system to send messages to QSYSOPR when a threshold value has been reached.

Once each day the following limits are checked against their alerting level. If the level is exceeded, a message is sent to the QSYSOPR message queue. Since these limits will prevent database activity from continuing if they are reached, you should take action to get the object's percent used for the limit below the alerting level. Reducing data by archiving it is one example of an action that could be taken.

Table 113. System limits that send alerting messages

Limit ID	Limit description	Maximum	Alerting Level	Alerting Cadence
15000	Maximum number of all rows in a partition	4,294,967,288	Greater than 90%	Once per day
15003	Maximum size of a table	1,869,169,767,219	Greater than 90%	Once per day
15104	Maximum number of variable-length segments	65,533	Greater than 90%	Once per day
15400	Maximum *MAX4GB Index Size	4,294,967,296	Greater than 90%	Once per day
15401	Maximum *MAX1TB Index Size	1,869,166,411,776	Greater than 90%	Once per day
15403	Maximum Encoded Vector Index Size	2,199,023,255,552	Greater than 90%	Once per day

The QSYSOPR message is formatted like this:

```
MYLIB/MYTABLE *FILE HAS CONSUMED MORE THAN 90% OF THE LIMIT:
15000-MAXIMUM NUMBER OF ALL ROWS (4008420999 OF 4294967288=93.33%).
REFER TO ibm.biz/DB2foriAlerts FOR MORE DETAIL.
```

SYSLIMTBL table

The SYSLIMTBL table contains information about limits that are being approached. It is maintained by DB2 for i.

This table is not authorized or managed like a typical DB2 for i catalog. By default, all users have authority to view and change the data within this table. If this table is removed or incompatibly altered, the IBM i operating system will automatically recreate it. The SYSLIMTBL table was designed to have as small a footprint as possible.

You can add AFTER INSERT or AFTER DELETE triggers to this table. This allows you to perform an action such as sending a notification when a limit is being logged to the table.

The following table describes the columns in the table. The schema is QSYS2.

Table 114. SYSLIMTBL table

Column Name	System Column Name	Data Type	Description
LAST_CHANGE_TIMESTAMP	LASTCHG	TIMESTAMP	The timestamp when this row was last changed.

Table 114. SYSLIMTBL table (continued)

Column Name	System Column Name	Data Type	Description
LIMIT_CATEGORY	CATEGORY	SMALLINT	The category of this limit. **0** Database **1** Journal **2** Security **3** Miscellaneous **4** Work management **5** File system **6** Save/restore **7** Cluster **8** Communications
LIMIT_TYPE	LIMTYPE	SMALLINT	The type of limit. **1** Object **2** Job **3** System **4** ASP
LIMIT_ID	LIMIT_ID	INTEGER	Unique identifier for this limit. Values are maintained in the SIZING_ID column in the QSYS2/SQL_SIZING table.
JOB_NAME	JOB_NAME	VARCHAR(28)	The name of the job that reported the current value.
USER_NAME	CURUSER	VARCHAR(10)	The name of the user in effect when the current value was updated.
CURRENT_VALUE	CURVAL	BIGINT	Reported value for this limit.
SYSTEM_SCHEMA_NAME	SYS_NAME	VARCHAR(10) Nullable	The library name for the object. If no library name, contains the null value.
SYSTEM_OBJECT_NAME	SYS_ONAME	VARCHAR(30) Nullable	The object name for this row. If no object name, contains the null value.
SYSTEM_TABLE_MEMBER	SYS_MNAME	VARCHAR(10) Nullable	The member name for an object limit specific to database members. Contains the null value if this row is not for a member limit.
OBJECT_TYPE	OBJTYPE	VARCHAR(7) Nullable	The IBM i object type when an object name has been logged in the SYSTEM_SCHEMA_NAME and SYSTEM_OBJECT_NAME columns. Contains the null value when no object name is specified.
ASP_NUMBER	ASPNUM	SMALLINT Nullable	Contains the ASP number related to this row. Contains the null value if there is no ASP number.
IFS_PATH_NAME	PATHNAME	DBCLOB(5000) CCSID 1200 Nullable	IFS path for the object. Contains the null value if there is no path.

SYSLIMITS view

The SYSLIMITS view contains information about limits. This view is built upon QSYS/SYSLIMTBL along with other system information. If a job is still active, it contains information about the job that logged the limit.

The following table describes the columns in the view. The schema is QSYS2.

Table 115. SYSLIMITS view

Column Name	System Column Name	Data Type	Description
LAST_CHANGE_TIMESTAMP	LASTCHG	TIMESTAMP	The timestamp when this row was last changed.
LIMIT_CATEGORY	CATEGORY	VARCHAR(15)	The category of this limit. • DATABASE • JOURNAL • SECURITY • MISCELLANEOUS • WORK MANAGEMENT • FILE SYSTEM • SAVE RESTORE • CLUSTER • COMMUNICATION • UNKNOWN
LIMIT_TYPE	TYPE	VARCHAR(7)	The type of limit. • OBJECT • JOB • SYSTEM • ASP • UNKNOWN
SIZING_NAME	SIZING_NAM	VARCHAR(128)	Name that corresponds to the sizing ID.
COMMENTS	COMMENTS	VARCHAR(2000) Nullable	Description of the limit.
USER_NAME	CURUSER	VARCHAR(10)	The name of the user in effect when this row was logged.
CURRENT_VALUE	CURVAL	BIGINT	Reported value for this limit.
MAXIMUM_VALUE	MAXVAL	DECIMAL(21,0) Nullable	Maximum value allowed for this limit.
JOB_NAME	JOB_NAME	VARCHAR(28)	The name of the job when this row was logged.
JOB_STATUS	JOB_STATUS	CHAR(10) Nullable	Status of the job.
ACTIVE_JOB_STATUS	AJSTATUS	CHAR(4) Nullable	The active status of the initial thread of the job.
RUN_PRIORITY	RUNPRI	INTEGER Nullable	The highest run priority allowed for any thread within this job.
SBS_NAME	SBS_NAME	CHAR(10) Nullable	Name of subsystem where job is running.
CPU_USED	CPU_USED	BIGINT Nullable	The amount of CPU time (in milliseconds) that has been currently used by this job.
TEMP_STORAGE_USED_MB	TEMPSTG	INTEGER Nullable	The amount of auxiliary storage (in megabytes) that is currently allocated to this job.
AUX_IO_REQUESTED	AUXIO	BIGINT Nullable	The number of auxiliary I/O requests performed by the job across all routing steps. This includes both database and nondatabase paging.

Table 115. SYSLIMITS view (continued)

Column Name	System Column Name	Data Type	Description
PAGE_FAULTS	PAGEFAULT	BIGINT Nullable	The number of times an active program referenced an address that was not in main storage during the current routing step of the specified job.
CLIENT_WRKSTNNAME	CLIENTWRK	CHAR(255) Nullable	Value of the SQL CLIENT_WRKSTNNAME special register.
CLIENT_APPLNAME	CLIENTAPP	CHAR(255) Nullable	Value of the SQL CLIENT_APPLNAME special register.
CLIENT_ACCTNG	CLIENTACT	CHAR(255) Nullable	Value of the SQL CLIENT_ACCTNG special register.
CLIENT_PROGRAMID	CLIENTPGM	CHAR(255) Nullable	Value of the SQL CLIENT_PROGRAMID special register.
CLIENT_USERID	CLIENTUSER	CHAR(255) Nullable	Value of the SQL CLIENT_USERID special register.
SQL_STATEMENT_TEXT	SQLSTMT	VARCHAR(10000) Nullable	Statement text of the last SQL statement to run or the SQL statement that is currently running.
SCHEMA_NAME	OBJ_SCHEMA	VARCHAR(128) Nullable	The SQL schema name for this object. If no schema name, contains the null value.
OBJECT_NAME	OBJ_NAME	VARCHAR(128) Nullable	The SQL name for the object. If no object name or if an SQL name could not be returned, contains the null value.
SYSTEM_SCHEMA_NAME	SYS_NAME	VARCHAR(10) Nullable	The library name for the object. If no library name, contains the null value.
SYSTEM_OBJECT_NAME	SYS_ONAME	VARCHAR(30) Nullable	The object name for this row. If no object name, contains the null value.
SYSTEM_TABLE_MEMBER	SYS_MNAME	VARCHAR(10) Nullable	The member name for an object limit specific to database members. Contains the null value if this row is not for a member limit.
IFS_PATH_NAME	PATHNAME	DBCLOB(5000) CCSID 1200 Nullable	IFS path for the object. Contains the null value if there is no path.
OBJECT_TYPE	OBJTYPE	VARCHAR(7) Nullable	The IBM i object type when an object name has been logged in the SYSTEM_SCHEMA_NAME and SYSTEM_OBJECT_NAME columns. Contains the null value when no object name is specified.

Table 115. SYSLIMITS view (continued)

Column Name	System Column Name	Data Type	Description
SQL_OBJECT_TYPE	SQLOBJTYPE	VARCHAR(9) Nullable	The SQL type of the object when an object name has been logged in the SYSTEM_SCHEMA_NAME and SYSTEM_OBJECT_NAME columns. Values are: • ALIAS • FUNCTION • INDEX • PACKAGE • PROCEDURE • ROUTINE • SEQUENCE • TABLE • TRIGGER • TYPE • VARIABLE • VIEW • XSR Contains the null value if the object is not an SQL object or when no object name is specified.
ASP_NUMBER	ASPNUM	SMALLINT Nullable	Contains the ASP number related to this row. Contains the null value if there is no ASP number.
LIMIT_ID	LIMIT_ID	INTEGER	Unique identifier for this limit. Values are maintained in the SIZING_ID column in the QSYS2/SQL_SIZING table.

QIBM_SYSTEM_LIMITS global variables
To prevent excess storage consumption within the SYS2/SYSLIMTBL table, DB2 for i will automatically delete (or prune) rows.

DB2 for i supplied global variables guide the pruning action. For each type of limit, there are two global variables. The pruning variable is used to choose how many of the most recently logged entries should be retained. The high point variable is used to choose how many of the highest consumption value entries should be retained.

The following are the names of the global variables and the limit that is shipped for each one. The schema is SYSIBMADM.

QIBM_SYSTEM_LIMITS_PRUNE_BY_ASP
100

QIBM_SYSTEM_LIMITS_PRUNE_BY_JOB
50

QIBM_SYSTEM_LIMITS_PRUNE_BY_OBJECT
20

QIBM_SYSTEM_LIMITS_PRUNE_BY_SYSTEM
100

QIBM_SYSTEM_LIMITS_SAVE_HIGH_POINTS_BY_ASP
25

QIBM_SYSTEM_LIMITS_SAVE_HIGH_POINTS_BY_JOB
5

QIBM_SYSTEM_LIMITS_SAVE_HIGH_POINTS_BY_OBJECT
5

QIBM_SYSTEM_LIMITS_SAVE_HIGH_POINTS_BY_SYSTEM
25

You can redefine any of the global variable values to change a limit on your system. After the variable is redefined, the new value takes effect after the next system IPL.

Use IBM i Navigator to generate SQL for the global variable and use the OR REPLACE option to recreate it with a different default. For example, to keep 50 of the most recently logged rows for all system types of limits, use the following SQL statement:

```
CREATE OR REPLACE VARIABLE SYSIBMADM.QIBM_SYSTEM_LIMITS_PRUNE_BY_SYSTEM
    INTEGER
    DEFAULT  50
```

Work Management Services

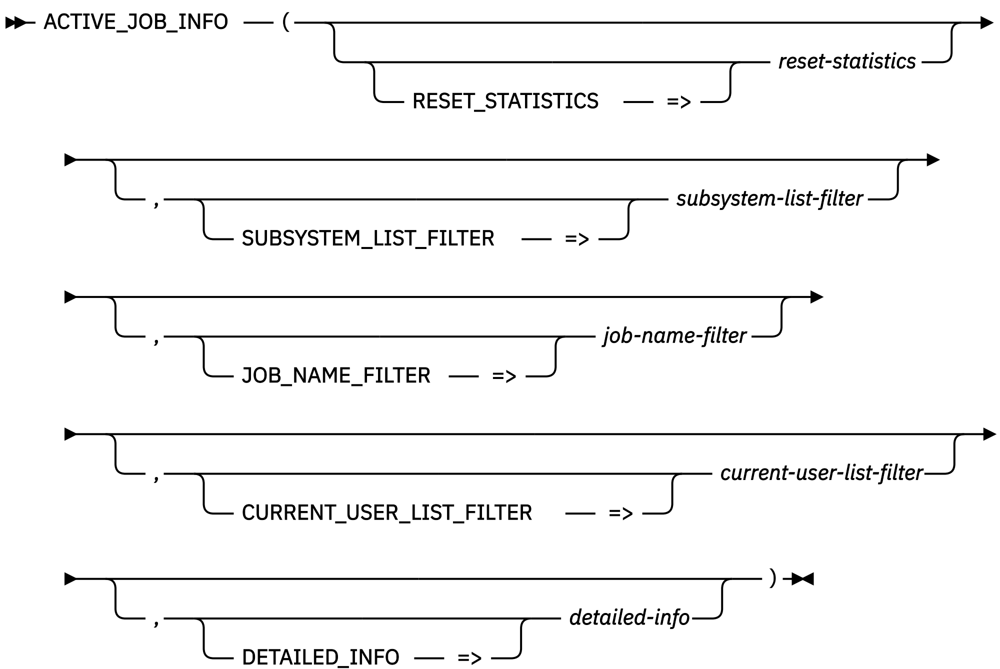
These views and functions provide system value and job information.

ACTIVE_JOB_INFO table function

The ACTIVE_JOB_INFO table function returns one row for every active job.

The information returned is similar to the detail seen from the Work with Active Jobs (WRKACTJOB) command. The ACTIVE_JOB_INFO table function has two uses:

1. To see details for all, or a subset of, active jobs. A subset of active jobs can be requested by using the optional filter parameters.
2. To measure elapsed statistics for active jobs. You can use an optional parameter to reset statistics, similar to the WRKACTJOB command F10 Restart Statistics function. Measurements will be calculated based on this new starting point.



The schema is QSYS2.

Authorization: None required to see general information or information about your own jobs.

For `DETAILED_INFO => ALL`:

- All users can see detailed column information for CLIENT_IP_ADDRESS, PAGE_FAULTS, JOB_ACTIVE_TIME, PRESTART_JOB_REUSE_COUNT, and PRESTART_JOB_MAX_USE_COUNT.
- A user with QIBM_DB_SQLADM or QIBM_DB_SYSMON function usage authority can see detailed column information that relates to SQL activity starting with the SQL_STATEMENT_TEXT column through the PSEUDO_CLOSED_CURSOR_COUNT column.
- For a user with *JOBCTL user special authority, all detailed column information is returned.

reset-statistics
A character or graphic string expression that contains a value of YES or NO.

If this parameter has a value of YES, statistics are reset such that the time of this query execution is used as the new baseline. Future invocations of ACTIVE_JOB_INFO within this connection will return statistical detail relative to the new baseline. If this parameter has a value of NO, statistics are not reset for the invocation unless the *subsystem-list-filter* or *job-name-filter* parameter values are different than the previous invocation. Changing the filter values will always cause statistics to be reset. If this parameter is not specified, the default is NO.

The first invocation of ACTIVE_JOB_INFO within a connection will always perform an implicit reset, regardless of whether a reset was explicitly requested.

subsystem-list-filter
A character or graphic string expression that contains a list of up to 25 subsystem names separated by exactly one comma. The filter determines which subsystems to use to return job information.

If this parameter is not specified, is an empty string, or is the null value, information for all subsystems is returned.

job-name-filter
A character or graphic string expression that contains an unqualified job name that determines the job information to be returned. The name can be a generic name.

The string can be one of the following special values:

*****
Only information for the current job is returned.

***ALL**
Information for all jobs is returned.

***CURRENT**
Information for all jobs with a job name that is the same as the current job is returned.

***SBS**
Information for all active subsystem monitors is returned.

***SYS**
Information for all active system jobs is returned. When using this value, the *subsystem-list-filter* must not be specified or must be the null value.

If this parameter is not specified, is an empty string, or is the null value, information for all jobs is returned.

current-user-list-filter
A character or graphic string expression that contains a list of up to 10 user profile names separated by exactly one comma. The filter determines which current user values to use to return job information.

If this parameter is not specified, is an empty string, or is the null value, information for all users is returned.

detailed-info
A character or graphic string expression that indicates the type of information to be returned.

ALL
Information for all the columns is returned.

NONE
Only the general information is returned for active jobs. This is the information in the columns prior to the JOB_DESCRIPTION_LIBRARY column. This is the default.

QTEMP
In addition to the general information for active jobs, the QTEMP_SIZE column is returned.

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 116. ACTIVE_JOB_INFO table function

Column Name	Data Type	Description
ORDINAL_POSITION	INTEGER	A unique number for each row.
JOB_NAME	VARCHAR(28)	The qualified job name.
INTERNAL_JOB_ID	BINARY(16)	The internal job identifier.
SUBSYSTEM	VARCHAR(10)	The name of the subsystem where the job is running. Contains the null value if the job is a system job.
SUBSYSTEM_LIBRARY_NAME	VARCHAR(10)	Library containing the subsystem description. Contains the null value if the job is a system job.
AUTHORIZATION_NAME	VARCHAR(10)	The user profile under which the initial thread is running at this time. For jobs that swap user profiles, this user profile name and the user profile that initiated the job can be different.
JOB_TYPE	VARCHAR(3)	Type of active job. **ASJ** Autostart **BCH** Batch **BCI** Batch Immediate **EVK** Started by a procedure start request **INT** Interactive **M36** Advanced 36 server job **MRT** Multiple requester terminal **PDJ** Print driver job **PJ** Prestart job **RDR** Spool reader **SBS** Subsystem monitor **SYS** System **WTR** Spool writer

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
FUNCTION_TYPE	VARCHAR(3)	The type of function described in the FUNCTION column. **CMD** The FUNCTION column contains the name of the command being run. **DLY** The initial thread of the job is processing a DLYJOB (Delay Job) command. The FUNCTION column contains a time that is the number of seconds the job is delayed (up to 999999 seconds), or the time when job is to resume processing (hh:mm:ss). **GRP** The FUNCTION column contains the group name of a suspended group job. **I/O** The job is a subsystem monitor that is performing input/output operations (I/O) to a work station for the sign-on display file. The FUNCTION column contains the name of the work station device. **IDX** The FUNCTION column contains the name of the file associated with an index rebuild operation. **JVM** The initial thread of the job is running a Java Virtual Machine. The FUNCTION column contains the name of the java class. **LOG** The FUNCTION column contains **QHST** to indicate history information is being logged to a database file. **MNU** The FUNCTION column contains the name of the menu. **MRT** The job is either a multiple requester terminal (MRT) job if JOB_TYPE is BCH, or it is an interactive job attached to an MRT job if JOB_TYPE is INT. For an MRT job, the FUNCTION column contains information in the following format: • CHAR(2): The number of requesters currently attached to the MRT job. • CHAR(1): Contains a / (slash). • CHAR(2): The maximum number of requesters. • CHAR(1): Contains a blank. • CHAR(3): The never-ending program (NEP) indicator. A value of NEP indicates a never-ending program. A value of blanks indicates that it is not a never-ending program. • CHAR(1): Contains a blank. For an interactive job attached to an MRT, the FUNCTION column contains the name of the MRT procedure. **PGM** The FUNCTION column contains the name of a program. **PRC** The FUNCTION column contains the name of a procedure. **USR** The FUNCTION column contains the user-specified function set with the Change Current Job (QWCCCJOB) API. Contains the null value if none of these values apply.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
FUNCTION	VARCHAR(10)	The last high-level function initiated by the initial thread. If FUNCTION_TYPE is not null, contains a value as defined by the FUNCTION_TYPE column. Otherwise, can contain one of the following values: **ADLACTJOB** Auxiliary storage is being allocated for the number of active jobs specified in the QADLACTJ system value. **ADLTOTJOB** Auxiliary storage is being allocated for the number of jobs specified in the QADLTOTJ system value. **CMDENT** The command entry display is being used. **COMMIT** The initial thread of the job is performing a commit operation. **DIRSHD** This job is running under the directory shadowing function. **DLTSPF** A spooled file is being deleted. **DUMP** A dump is in process. **JOBIDXRCY** A damaged job index is being recovered. **JOBLOG** A job log is being produced. **JOBLOGQRCY** The job log server queue is being recovered or rebuilt. **PASSTHRU** The job is a pass-through job. **RCLSPLSTG** Empty spooled database members are being deleted. **ROLLBACK** The initial thread of the job is performing a rollback operation. **SPLCLNUP** A cleanup of jobs on job queues and spooled files is being performed. Contains the null value if a logged function has not been performed.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
JOB_STATUS	VARCHAR(4)	The status of the initial thread of the job. The following list contains some of the most common values. For a complete list of values, see Work Management API Attribute Descriptions in Application Programming Interfaces **CMNW** Waiting for the completion of an I/O operation to a communications device. **CNDW** Waiting on handle-based condition. **DEQW** Waiting for completion of a dequeue operation. **DLYW** Due to the Delay Job (DLYJOB) command, the initial thread of the job is delayed while it waits for a time interval to end, or for a specific delay end time. **DSPW** Waiting for input from a work station display. **END** The job has been ended with the *IMMED option, or its delay time has ended with the *CNTRLD option. **EOJ** Ending for a reason other than running the End Job (ENDJOB) or End Subsystem (ENDSBS) command. **EVTW** Waiting for an event. **HLD** The job is being held. **JVAW** Waiting for completion of a Java program operation. **LCKW** Waiting for a lock. **LSPW** Waiting for a lock space to be attached. **MSGW** Waiting for a message from a message queue. **MTXW** Waiting for a mutex. **PSRW** A prestart job waiting for a program start request. **RUN** Job is currently running. **SEMW** Waiting for a semaphore. **THDW** Waiting for another thread to complete an operation.
MEMORY_POOL	VARCHAR(9)	The identifier of the system-related pool from which the job's main storage is allocated.
RUN_PRIORITY	INTEGER	The run priority of the job.
THREAD_COUNT	INTEGER	The number of active threads in the job.
TEMPORARY_STORAGE	INTEGER	The amount of temporary storage, in megabytes, that is currently allocated to this job.
CPU_TIME	DECIMAL(20,0)	The total processing unit time used by the job, in milliseconds.
TOTAL_DISK_IO_COUNT	DECIMAL(20,0)	The total number of disk I/O operations performed by the job across all routing steps. This is the sum of the asynchronous and synchronous disk I/O.
ELAPSED_INTERACTION_COUNT	INTEGER	The number of interactions. This is the number of operator interactions during the measurement time interval. Contains the null value if the job is not interactive.
ELAPSED_TOTAL_RESPONSE_TIME	INTEGER	The total response time over the measurement time interval, in seconds. Contains the null value if the job is not interactive.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
ELAPSED_TOTAL_DISK_IO_COUNT	DECIMAL(20,0)	The number of disk I/O operations performed by the job during the measurement time interval. This is the sum of the asynchronous and synchronous disk I/O.
ELAPSED_ASYNC_DISK_IO_COUNT	DECIMAL(20,0)	The number of asynchronous (physical) disk I/O operations performed by the job during the measurement time interval. This value is the sum of the asynchronous database and nondatabase reads and writes.
ELAPSED_SYNC_DISK_IO_COUNT	DECIMAL(20,0)	The number of synchronous (physical) disk I/O operations performed by the job during the measurement time interval. This value is the sum of the synchronous database and nondatabase reads and writes.
ELAPSED_CPU_PERCENTAGE	DECIMAL(10,1)	The percent of processing unit time attributed to this job during the measurement time interval.
ELAPSED_CPU_TIME	DECIMAL(20,0)	The total CPU time spent during the measurement time interval, in milliseconds.
ELAPSED_PAGE_FAULT_COUNT	DECIMAL(20,0)	The number of times an active program referenced an address that is not in main storage for the specified job during the measurement time interval.
JOB_END_REASON	VARCHAR(60)	Reason the job is ending. Contains one of the following values: • JOB ENDING IN NORMAL MANNER • JOB ENDED WHILE IT WAS STILL ON A JOB QUEUE • SYSTEM ENDED ABNORMALLY • JOB ENDING NORMALLY AFTER A CONTROLLED END WAS REQUESTED • JOB ENDING IMMEDIATELY • JOB ENDING ABNORMALLY • JOB ENDED DUE TO THE CPU LIMIT BEING EXCEEDED • JOB ENDED DUE TO THE STORAGE LIMIT BEING EXCEEDED • JOB ENDED DUE TO THE MESSAGE SEVERITY LEVEL BEING EXCEEDED • JOB ENDED DUE TO THE DISCONNECT TIME INTERVAL BEING EXCEEDED • JOB ENDED DUE TO THE INACTIVITY TIME INTERVAL BEING EXCEEDED • JOB ENDED DUE TO A DEVICE ERROR • JOB ENDED DUE TO A SIGNAL • JOB ENDED DUE TO AN UNHANDLED ERROR Contains the null value if job is not currently ending.
SERVER_TYPE	VARCHAR(30)	The type of server represented by the job. See Server table for a list of server type values. Contains the null value if the job is not part of a server.
ELAPSED_TIME	DECIMAL(20,3)	The time that has elapsed, in seconds, between the measurement start time and the current system time.
Values for the following columns are returned when the DETAILED_INFO parameter is ALL. Otherwise, the columns will contain the null value.		
JOB_DESCRIPTION_LIBRARY	VARCHAR(10)	The name of the library containing the job description. Contains the null value if the job has no job description.
JOB_DESCRIPTION	VARCHAR(10)	The name of the job description used for this job. Contains the null value if the job has no job description.
JOB_QUEUE_LIBRARY	VARCHAR(10)	The name of the library containing the job queue. Contains the null value if the job is not a batch job that was started from a job queue.
JOB_QUEUE	VARCHAR(10)	The name of the job queue that the job was on. Contains the null value if the job is not a batch job that was started from a job queue.
OUTPUT_QUEUE_LIBRARY	VARCHAR(10)	The name of the library that contains the default output queue. Contains the null value if the job has no default output queue.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
OUTPUT_QUEUE	VARCHAR(10)	The name of the default output queue that is used for spooled output produced by this job. The default output queue is only used by spooled printer files that specify *JOB for the output queue. Contains the null value if the job has no default output queue.
CCSID	INTEGER	The coded character set identifier (CCSID) used for this job.
DEFAULT_CCSID	INTEGER	The default coded character set identifier used for this job.
SORT_SEQUENCE_LIBRARY	VARCHAR(10)	The name of the library that contains the sort sequence table. Contains the null value if no sort sequence table is defined for this job or if SORT_SEQUENCE is a special value.
SORT_SEQUENCE	VARCHAR(10)	The name of the sort sequence table associated with this job. Contains the null value if no sort sequence table is defined for this job.
LANGUAGE_ID	CHAR(3)	The language identifier associated with this job.
DATE_FORMAT	CHAR(4)	The date format used for this job. ***DMY** Day, month, year format. ***JUL** Julian format (year and day). ***MDY** Month, day, year format. ***YMD** Year, month, day format.
DATE_SEPARATOR	CHAR(1)	The date separator used for this job.
TIME_SEPARATOR	CHAR(1)	The time separator used for this job.
DECIMAL_FORMAT	VARCHAR(6)	The decimal format used for this job. ***BLANK** Uses a period for a decimal point, a comma for a 3-digit grouping character, and zero-suppress to the left of the decimal point. **J** Uses a comma for a decimal point and a period for a 3-digit grouping character. The zero-suppression character is in the second position (rather than the first) to the left of the decimal notation. Balances with zero values to the left of the comma are written with one leading zero (0,04). The J entry also overrides any edit codes that might suppress the leading zero. **I** Uses a comma for a decimal point, a period for a 3-digit grouping character, and zero-suppress to the left of the decimal point.
TIMEZONE_DESCRIPTION	VARCHAR(10)	The name of the time zone description that is used to calculate local job time.
TIMEZONE_CURRENT_OFFSET	INTEGER	The offset, in minutes, used to calculate local job time. This value has been adjusted for Daylight Saving Time, if necessary.
TIMEZONE_FULL_NAME	VARCHAR(50)	The full, or long, name for the time zone. This column returns either the standard or Daylight Saving Time full name depending on whether or not Daylight Saving Time is in effect. Contains the null value if the time zone description uses a message to specify the current full name and the message cannot be retrieved.
TIMEZONE_ABBREVIATED_NAME	VARCHAR(10)	The abbreviated, or short, name for the time zone. This column returns either the standard or Daylight Saving Time abbreviated name depending on whether or not Daylight Saving Time is in effect. Contains the null value if the time zone description uses a message to specify the current abbreviated name and the message cannot be retrieved.
JOB_ENTERED_SYSTEM_TIME	TIMESTAMP(0)	The timestamp for when the job was placed on the system.
JOB_ACTIVE_TIME	TIMESTAMP(0)	The timestamp for when the job began to run on the system.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
CLIENT_IP_ADDRESS	VARCHAR(45)	Client IP address, in IPv4 format, being used by the job. Contains the null value when no client IP address exists or the job is using IPv6.
JOB_USER_IDENTITY_SETTING	VARCHAR(11)	The method by which the job user identity was set. **APPLICATION** The job user identity was explicitly set by an application using one of the Set Job User Identity APIs, QWTSJUID or QwtSetJuid(). The job may be running either single threaded or multithreaded. **DEFAULT** The job is currently running single threaded and the job user identity is the name of the user profile under which the job is currently running. **SYSTEM** The job is currently running multithreaded and the job user identity was implicitly set by the system when the job became multithreaded. It was set to the name of the user profile that the job was running under when it became multithreaded.
JOB_USER_IDENTITY	VARCHAR(10)	The user profile name by which the job is known to other jobs on the system. The job user identity is used for authorization checks when other jobs on the system attempt to operate against the job. Contains the null value if the user profile no longer exists.
DBCS_CAPABLE	VARCHAR(3)	Whether the job is DBCS-capable. **NO** The job is not DBCS-capable. **YES** The job is DBCS-capable.
SIGNAL_STATUS	VARCHAR(3)	Whether the job is enabled to receive signals from another job or the system. **NO** The job is not enabled for signals. This job cannot receive signals from another job or the system. **YES** The job is enabled for signals. This job can receive signals from another job or the system.
MESSAGE_REPLY	VARCHAR(3)	Whether the job is waiting for a reply to a specific message. **NO** The job is not waiting for a reply to a message. **YES** The job is waiting for a reply to a message. Contains the null value if the job is not in message wait status.
END_STATUS	VARCHAR(3)	Whether the system issued a controlled cancellation. **NO** The system, subsystem, or job is not canceled. **YES** The system, the subsystem in which the job is running, or the job itself is canceled.
CANCEL_KEY	VARCHAR(3)	Whether the user pressed the cancel key. **NO** The user did not press the cancel key. **YES** The user pressed the cancel key.
EXIT_KEY	VARCHAR(3)	Whether the user pressed the exit key. **NO** The user did not press the exit key. **YES** The user pressed the exit key.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
MAXIMUM_ACTIVE_THREADS	INTEGER	The maximum number of threads that a job can run with at any time. If multiple threads are initiated simultaneously, this value may be exceeded. If this maximum value is exceeded, the excess threads will be allowed to run to their normal completion. Initiation of additional threads will be inhibited until the maximum number of threads in the job drops below this maximum value. Contains the null value if there is no maximum.
SYSTEM_POOL_ID	INTEGER	The identifier of the system-related pool from which main storage is currently being allocated for the job's initial thread. These identifiers are not the same as those specified in the subsystem description, but are the same as the system pool identifiers shown on the system status display. If a thread reaches its time-slice end, the pool the thread is running in can be switched based on the job's time-slice end pool value. The current system pool identifier returned will be the actual pool in which the initial thread of the job is running.
POOL_NAME	VARCHAR(10)	The name of the memory pool in which the job started running. The name may be a number, in which case it is a private pool associated with a subsystem. Can contain one of the following special values: ***BASE** This job is running in the base system pool, which can be shared with other subsystems. ***INTERACT** This job is running in the shared pool used for interactive work. ***MACHINE** This job is running in the machine pool. ***SHRPOOL1 - *SHRPOOL60** This job is running in the identified shared pool. ***SPOOL** This job is running in the shared pool for spooled writers.
QTEMP_SIZE	INTEGER	The amount of storage, in megabytes, used by objects in the job's temporary library (QTEMP). Objects that are locked, damaged, or not authorized are not included. Contains the null value if the size cannot be returned.
PEAK_TEMPORARY_STORAGE	INTEGER	The maximum amount of auxiliary storage, in megabytes, that the job has used.
DEFAULT_WAIT	INTEGER	The default maximum time, in seconds, that a thread in the job waits for a system instruction, such as a LOCK machine interface (MI) instruction, to acquire a resource. Contains the null value if there is no maximum.
MAXIMUM_PROCESSING_TIME_ ALLOWED	INTEGER	The maximum processing unit time, in milliseconds, that the job can use. If the job consists of multiple routing steps, this is the maximum processing unit time that the current routing step can use. If the maximum time is exceeded, the job is held. Contains the null value if no maximum amount of processing unit time has been defined.
MAXIMUM_TEMPORARY_STORAGE_ ALLOWED	INTEGER	The maximum amount of auxiliary storage, in megabytes, that the job can use. If the job consists of multiple routing steps, this is the maximum temporary storage that the routing step can use. This temporary storage is used for storage required by the program itself and by implicitly created internal system objects used to support the routing step. (It does not include storage for objects in the QTEMP library.) If the maximum temporary storage is exceeded, the job is held. This does not apply to the use of permanent storage, which is controlled through the user profile. Contains the null value if no maximum amount of temporary storage has been defined.
TIME_SLICE	INTEGER	The maximum amount of processor time, in milliseconds, given to each thread in this job before other threads in this job and in other jobs are given the opportunity to run. The time slice establishes the amount of time needed by a thread in this job to accomplish a meaningful amount of processing. At the end of the time slice, the thread might be put in an inactive state so that other threads can become active in the storage pool. Values range from 8 through 9999999.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
PAGE_FAULTS	BIGINT	The number of times an active program referenced an address that was not in main storage during the current routing step of the specified job.
TOTAL_RESPONSE_TIME	BIGINT	The total amount of response time for the initial thread, in milliseconds. This value does not include the time used by the machine, by the attached input/output (I/O) hardware, and by the transmission lines for sending and receiving data. Returns zero for jobs that have no interactions. A value of -1 is returned if the field is not large enough to hold the actual result.
INTERACTIVE_TRANSACTIONS	INTEGER	The count of operator interactions, such as pressing the Enter key or a function key. Returns zero for jobs that have no interactions.
DATABASE_LOCK_WAITS	INTEGER	The number of times that the initial thread had to wait to obtain a database lock.
NON_DATABASE_LOCK_WAITS	INTEGER	The number of times that the initial thread had to wait to obtain a nondatabase lock.
INTERNAL_MACHINE_ LOCK_WAITS	INTEGER	The number of times that the initial thread had to wait to obtain an internal machine lock.
DATABASE_LOCK_WAIT_TIME	INTEGER	The cumulative amount of time, in milliseconds, that the initial thread has had to wait to obtain database locks.
NON_DATABASE_LOCK_WAIT_TIME	INTEGER	The cumulative amount of time, in milliseconds, that the initial thread has had to wait to obtain nondatabase locks.
INTERNAL_MACHINE_LOCK_ WAIT_TIME	INTEGER	The cumulative amount of time, in milliseconds, that the initial thread has had to wait to obtain internal machine locks.
SQL_STATEMENT_TEXT	VARCHAR(10000)	Statement text of the last SQL statement to run or the SQL statement that is currently running. The statement text will be truncated if it is longer than the column. Contains the null value if no SQL statement has been run.
SQL_STATEMENT_STATUS	VARCHAR(8)	The status of SQL within this job. **ACTIVE** An SQL statement is currently running **COMPLETE** At least one SQL statement has run and has completed Contains the null value if no SQL statement has been run.
SQL_STATEMENT_START_TIMESTAMP	TIMESTAMP	The timestamp of the execution start for an active SQL statement. Contains the null value if there is no active SQL statement.
SQL_STATEMENT_NAME	VARCHAR(128)	The name of the SQL statement. Contains the null value when the SQL statement has no name.
SQL_STATEMENT_LIBRARY_NAME	VARCHAR(10)	The library name for the SQL statement object. Contains the null value when the SQL statement name is null or when the SQL statement does not exist within a permanent object.
SQL_STATEMENT_OBJECT_NAME	VARCHAR(10)	The name of the object which contains the last SQL statement executed in the job. When the current SQL statement belongs to an SQL function or an SQL procedure, the object name will be the external program name. Contains the null value when the SQL statement name is null or when the SQL statement does not exist within a permanent object.
SQL_STATEMENT_OBJECT_TYPE	VARCHAR(7)	The type of object containing the current SQL statement. ***PGM** The current SQL statement resides within a program. ***SQLPKG** The current SQL statement resides within an SQL package. ***SRVPGM** The current SQL statement resides within a service program. Contains the null value when the SQL statement object name is null.
QUERY_OPTIONS_LIBRARY_NAME	VARCHAR(10)	The name of the QAQQINI options library in use for this job.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
SQL_ACTIVATION_GROUP_COUNT	INTEGER	The number of activation groups, current and ended, that have executed SQL statements for the job. Contains the null value if no SQL statement has been run.
SQL_DESCRIPTOR_COUNT	BIGINT	The number of SQL descriptors that are active for the job. Contains the null value if no SQL descriptors are active for the job.
SQL_LOB_LOCATOR_COUNT	INTEGER	The number of LOB locators that are active for the job. Contains the null value if no LOB locators are active for the job.
CLI_HANDLE_COUNT	BIGINT	The number of SQL Call Level Interface (CLI) handles that are active for the job. This count includes CLI statement handles, descriptor handles, environment handles, and connection handles. Contains the null value if no CLI handles are active for the job.
SQL_SERVER_MODE	VARCHAR(3)	Indicates whether the job is configured to use SQL Server Mode. **NO** The job is not configured to use SQL Server Mode. **YES** The job is configured to use SQL Server Mode.
CLIENT_ACCTNG	VARCHAR(255)	Value of the SQL CURRENT CLIENT_ACCTNG special register. The value can be null. For more information, see CURRENT CLIENT_ACCTNG.
CLIENT_APPLNAME	VARCHAR(255)	Value of the SQL CURRENT CLIENT_APPLNAME special register. The value can be null. For more information, see CURRENT CLIENT_APPLNAME.
CLIENT_PROGRAMID	VARCHAR(255)	Value of the SQL CURRENT CLIENT_PROGRAMID special register. The value can be null. For more information, see CURRENT CLIENT_PROGRAMID.
CLIENT_USERID	VARCHAR(255)	Value of the SQL CURRENT CLIENT_USERID special register. The value can be null. For more information, see CURRENT CLIENT_USERID.
CLIENT_WRKSTNNAME	VARCHAR(255)	Value of the SQL CURRENT CLIENT_WRKSTNNAME special register. The value can be null. For more information, see CURRENT CLIENT_WRKSTNNAME.
ROUTINE_TYPE	CHAR(1)	For a routine defined using SQL, the type of the currently executing routine. **F** Function **P** Procedure Contains the null value if there is no SQL routine currently executing.
ROUTINE_SCHEMA	VARCHAR(128)	For a routine defined using SQL, the schema name of the currently executing routine. Contains the null value if there is no SQL routine currently executing.
ROUTINE_SPECIFIC_NAME	VARCHAR(128)	For a routine defined using SQL, the name of the currently executing routine. Contains the null value if there is no SQL routine currently executing.
CLIENT_PORT	INTEGER	The port number used by the current client to communicate with the server. Contains the null value if the target job does not correspond to a connection formed using the TCP/IP protocol.
CLIENT_HOST	VARCHAR(255)	The host name used by the current client to communicate with the server. Contains the null value if the target job does not correspond to a connection formed using the TCP/IP protocol.
INTERFACE_NAME	VARCHAR(127)	The client database interface name. Contains the null value if there is no client database interface name.
INTERFACE_TYPE	VARCHAR(63)	The client database interface type. Contains the null value if there is no client database interface type.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
INTERFACE_LEVEL	VARCHAR(63)	The client database interface level in the following form: "VVRRMMFP". VV - Version RR - Release MM - Modification level FP - Fix pack level (only applicable for certain interfaces). Contains the null value if there is no client database interface level.
SERVER_MODE_CONNECTING_JOB	VARCHAR(28)	The qualified job name of the job that established the SQL Server Mode connection. If the job name is QSQSRVR, then the qualified job name of the connecting job is returned. Contains the null value if the job name is not QSQSRVR or JOB_STATUS is PSRW.
SERVER_MODE_CONNECTING_THREAD	BIGINT	If the job name is QSQSRVR and the server mode job is in use, the thread identifier of the last thread to use this connection is returned. When SQL_STATEMENT_STATUS is COMPLETE, this application thread identifier might no longer exist. Contains the null value if the job name is not QSQSRVR or JOB_STATUS is PSRW.
PRESTART_JOB_REUSE_COUNT	INTEGER	The number of times the prestart job has been used. The prestart job reuse count is incremented when a disconnect is processed for a prestart job. When the prestart job reuse count exceeds the prestart job maximum number of uses, the job is ended. Contains the null value if the job is not a prestart job.
PRESTART_JOB_MAX_USE_COUNT	INTEGER	The maximum number of times the prestart job can be used before it is ended. A value of -1 is returned for *NOMAX. Contains the null value if the job is not a prestart job.
AVAILABLE_RESULT_SETS	INTEGER	The current count of unconsumed SQL result sets for the job. Contains the null value if the job has no unconsumed SQL result sets.
UNCONSUMED_RESULT_SETS	INTEGER	The cumulative count of unconsumed SQL result sets that were discarded for the job. Contains the null value if the job has no unconsumed SQL result sets that have been discarded.
OPEN_CURSOR_COUNT	INTEGER	The number of SQL cursors that are currently open for the job. Contains the null value if no SQL cursors are currently open for the job.
FULL_OPEN_CURSOR_COUNT	BIGINT	The total number of SQL cursors that have been full opened for the life of the job. Contains the null value if no SQL cursors have been full opened during the life of the job.
PSEUDO_OPEN_CURSOR_COUNT	BIGINT	The total number of SQL cursors that have been pseudo opened for the life of the job. Pseudo opens are also known as reused SQL cursors. Contains the null value if no SQL cursors have been pseudo opened during the life of the job.
PSEUDO_CLOSED_CURSOR_COUNT	INTEGER	The active number of pseudo closed SQL cursors within the job. Pseudo closed cursors are cursors that have been closed by the application, but remain open within the database. A pseudo closed cursor may be reused when the same query is executed many times, resulting in a performance improvement on the open. Conversely, accumulating too many pseudo closed cursors within the job can have a negative impact on the storage footprint of the job. Contains the null value if no SQL cursors are pseudo closed.
CQE_CURSOR_COUNT	INTEGER	The number of cursors using CQE for this job. This includes SQL cursors (both fully opened and pseudo closed) and cursors used to implement native database queries. Contains the null value if no cursors have used CQE for this job.
CQE_CURSOR_STORAGE	INTEGER	The amount of storage, in megabytes, used by cursors using CQE for this job. Contains the null value if no cursors have used CQE for this job.

Table 116. ACTIVE_JOB_INFO table function (continued)

Column Name	Data Type	Description
SQE_CURSOR_COUNT	INTEGER	The number of cursors using SQE for this job. This includes SQL cursors (both fully opened and pseudo closed) and cursors used to implement native database queries. Contains the null value if no cursors have used SQE for this job.
SQE_CURSOR_STORAGE	INTEGER	The amount of storage, in megabytes, used by cursors using SQE for this job. Contains the null value if no cursors have used SQE for this job.
LARGEST_QUERY_SIZE	INTEGER	The amount of storage, in megabytes, used by the SQE cursor that used the most storage for this job. Contains the null value if no cursors have used SQE for this job.
QRO_HASH	VARCHAR(8)	An internally generated identifier for the SQE query referred to in the LARGEST_QUERY_SIZE column. The QRO hash surfaces within Visual Explain and from Show Statements exploration of the SQL Plan Cache and SQL Plan Cache Snapshots. Contains the null value if no cursors have used SQE for this job.

Examples

- **Example 1:** Looking at only QZDASOINIT jobs, find the top 10 consumers of Elapsed I/O.

```
SELECT JOB_NAME, AUTHORIZATION_NAME, ELAPSED_TOTAL_DISK_IO_COUNT, ELAPSED_CPU_PERCENTAGE
FROM TABLE(QSYS2.ACTIVE_JOB_INFO(
            JOB_NAME_FILTER => 'QZDASOINIT',
            SUBSYSTEM_LIST_FILTER => 'QUSRWRK')) X
ORDER BY ELAPSED_TOTAL_DISK_IO_COUNT DESC
FETCH FIRST 10 ROWS ONLY;
```

  Note: The data in the ELAPSED_xxx columns is updated upon each re-execution of the query. Elapsed data will not get returned the first time a query is run for ACTIVE_JOB_INFO for a connection. See the *reset-statistics* parameter for details.

- **Example 2:** Find the active jobs using the most temporary storage. Include the most recently executed SQL statement for each target job.

```
SELECT JOB_NAME, AUTHORIZATION_NAME, TEMPORARY_STORAGE, SQL_STATEMENT_TEXT
FROM TABLE (QSYS2.ACTIVE_JOB_INFO(DETAILED_INFO => 'ALL')) X
    WHERE JOB_TYPE <> 'SYS'
ORDER BY TEMPORARY_STORAGE DESC;
```

JOB_DESCRIPTION_INFO view

The JOB_DESCRIPTION_INFO view returns information about job descriptions.

The values returned for the columns in the view are closely related to the values returned by the Display Job Description (DSPJOBD) CL command and the Retrieve Job Description Information (QWDRJOBD) API.

Authorization: Rows will be returned for job descriptions when the caller has *EXECUTE authority to the job description library and *USE authority to the job description.

The following table describes the columns in the view. The system name is JOBD_INFO. The schema is QSYS2.

Table 117. JOB_DESCRIPTION_INFO view

Column Name	System Column Name	Data Type	Description
JOB_DESCRIPTION_LIBRARY	JOBDLIB	VARCHAR(10)	The name of the library in which the job description resides.
JOB_DESCRIPTION	JOBD	VARCHAR(10)	The name of the job description about which information is being returned.
AUTHORIZATION_NAME	USER_NAME	VARCHAR(10)	The name of the user profile associated with this job description. Can contain the following special value: ***RQD** A user name is required to use the job description.

Table 117. JOB_DESCRIPTION_INFO view (continued)

Column Name	System Column Name	Data Type	Description
JOB_DATE	JOB_DATE	DATE Nullable	The date that will be assigned to jobs using this job description when they are started. Contains the null value if this job will use the QDATE system value.
ACCOUNTING_CODE	ACGCDE	VARCHAR(15)	An identifier assigned to jobs that use this job description. This code is used to collect system resource use information. Can contain the following special value: ***USRPRF** The accounting code used for jobs using this job description is obtained from the job's user profile.
ROUTING_DATA	RTGDTA	VARCHAR(80)	The routing data that is used with this job description to start jobs. Can contain one of the following special values: **QCMDI** The default routing data QCMDI is used by the IBM-supplied interactive subsystem to route the job to the IBM-supplied control language processor QCMD in the QSYS library. ***RQSDTA** Up to the first 80 characters of the request data specified in the request data field are used as the routing data for the job.
REQUEST_DATA	RQSDTA	VARCHAR(256) Nullable	The request data that is placed as the last entry in the job's message queue for jobs that use this job description. Can contain the following special value: ***RTGDTA** The data specified in the routing data parameter is placed as the last entry in the job's message queue. Contains the null value if no request data is placed in the job's message queue.
LIBRARY_LIST_COUNT	LIBL_COUNT	INTEGER	The number of libraries in the user portion of the initial library list.
LIBRARY_LIST	LIBL	VARCHAR(2750) Nullable	The initial library list that is used for jobs that use this job description. Only the libraries in the user portion of the library list are included. The list is an array of 11 character entries. Each entry contains a ten character name followed by one blank. Can contain the following special value: ***SYSVAL** The jobs using this job description will use the library list specified by the QUSRLIBL system value. Contains the null value is there is no initial library list.
JOB_SWITCHES	SWITCHES	CHAR(8)	The initial settings for a group of eight job switches used by jobs that use this job description. These switches can be set or tested in a program and used to control a program's flow. The possible values are '0' (off) and '1' (on).
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	The user text, if any, used to briefly describe the job description. Contains the null value is there is no descriptive text.
JOB_QUEUE_LIBRARY	JOBQLIB	VARCHAR(10)	The library of the job queue into which batch jobs using this job description are placed.
JOB_QUEUE	JOBQ	VARCHAR(10)	The name of the job queue into which batch jobs using this job description are placed.
JOB_QUEUE_PRIORITY	JOBQ_PRI	SMALLINT	The scheduling priority of each job that uses this job description. The highest priority is 1 and the lowest priority is 9.
HOLD_ON_JOB_QUEUE	JOBQ_HOLD	VARCHAR(4)	Whether jobs using this job description are put on the job queue with a status of held. ***NO** Jobs using this job description are not put on the job queue as held. ***YES** Jobs using this job description are put on the job queue as held.

Table 117. JOB_DESCRIPTION_INFO view (continued)

Column Name	System Column Name	Data Type	Description
OUTPUT_QUEUE_LIBRARY	OUTQLIB	VARCHAR(10) Nullable	The name of the library in which the output queue resides. Contains the null value if OUTPUT_QUEUE is a special value.
OUTPUT_QUEUE	OUTQ	VARCHAR(10)	The name of the default output queue that is used for spooled output produced by jobs that use this job description. Can contain one of the following special values: ***DEV** The output queue with the same name as the printer device for this job description is used. ***USRPRF** The output queue name for jobs using this job description is obtained from the user profile of the job at the time the job is started. ***WRKSTN** The output queue name is obtained from the device description from which this job is started.
OUTPUT_QUEUE_PRIORITY	OUTQ_PRI	SMALLINT	The output priority for spooled files that are produced by jobs using this job description. The highest priority is 1, and the lowest priority is 9.
SPOOLED_FILE_ACTION	SPOOL_ACT	VARCHAR(7)	Specifies whether spooled files can be accessed through job interfaces once a job has completed its normal activity. ***DETACH** Spooled files are detached from the job when the job completes its activity. ***KEEP** When the job completes its activity, as long as at least one spooled file for the job exists in the system auxiliary storage pool (ASP 1) or in a basic user ASP (ASPs 2-32), the spooled files are kept with the job and the status of the job is updated to indicate that the job has completed. If all remaining spooled files for the job are in independent ASPs (ASPs 33-255), the spooled files will be detached from the job and the job will be removed from the system. ***SYSVAL** The jobs using this job description will take the spooled file action specified by the QSPLFACN system value.
PRINTER_DEVICE	DEV_NAME	VARCHAR(10)	The name of the printer device that is used for all spooled files created by jobs that use this job description. Can contain one of the following special values: ***SYSVAL** The value in the system value QPRTDEV at the time the job is started is used as the printer device name. ***USRPRF** The printer device name is obtained from the user profile of the job at the time the job is started. ***WRKSTN** The printer device name is obtained from the work station where the job was started.
PRINT_TEXT	PRINT_TEXT	VARCHAR(30) Nullable	The line of text that is printed at the bottom of each page of printed output for jobs using this job description. Can contain the following special value: ***SYSVAL** The value in the system value QPRTTXT is used for jobs using this job description. Contains the null value if there is no text to print.
JOB_MESSAGE_QUEUE _MAXIMUM_SIZE	MSGQ_MAX	SMALLINT Nullable	The maximum size (in megabytes) of the job message queue. The possible values are 2 to 64. Contains the null value if the maximum size is set by system value QJOBMSGQMX at the time the job is started.

Table 117. JOB_DESCRIPTION_INFO view (continued)

Column Name	System Column Name	Data Type	Description
JOB_MESSAGE_QUEUE_FULL _ACTION	MSGQ_FULL	VARCHAR(8)	The action taken when the job message queue becomes full. ***NOWRAP** When the message queue becomes full, do not wrap. This action will cause the job to end. ***PRTWRAP** When the message queue becomes full, wrap the job queue and print the messages that are being overlaid. ***SYSVAL** The value is specified by the system value QJOBMSGQFL. ***WRAP** When the message queue becomes full, wrap to the beginning and start filling again.
SYNTAX_CHECK_SEVERITY	SYNTAX	SMALLINT Nullable	Whether requests placed on the job's message queue are checked for syntax as CL commands, and the message severity that causes a syntax error to end processing of a job. The possible values are: **0-99** Specifies the lowest message severity that causes a running job to end. The request data is checked for syntax as CL commands, and, if a syntax error occurs that is greater than or equal to the error message severity specified here, the running of the job that contains the erroneous command is suppressed. Contains the null value if the request data is not checked for syntax as CL commands.
JOB_END_SEVERITY	JOB_ENDSEV	SMALLINT	The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message whose severity is equal to or greater than this value to the request processing program. The possible values are from 0 through 99.
JOBLOG_OUTPUT	JOBLOG_OUT	VARCHAR(10)	How the job log will be produced when the job completes. This does not affect job logs produced when the message queue is full and the job message queue full action specifies *PRTWRAP. Messages in the job message queue are written to a spooled file, from which the job log can be printed, unless the Control Job Log Output (QMHCTLJL) API was used in the job to specify that the messages in the job log are to be written to a database file. The job log output value can be changed at any time until the job log has been produced or removed. To change the job log output value for a job, use the Change Job (QWTCHGJB) API or the Change Job (CHGJOB) command. The job log can be displayed at any time until the job log has been produced or removed. To display the job log, use the Display Job Log (DSPJOBLOG) command. The job log can be removed when the job has completed and the job log has not yet been produced or removed. To remove the job log, use the Remove Pending Job Log (QWTRMVJL) API or the End Job (ENDJOB) command. The possible values are: ***JOBEND** The job log will be produced by the job itself. If the job cannot produce its own job log, the job log will be produced by a job log server. For example, a job does not produce its own job log when the system is processing a Power Down System (PWRDWNSYS) command. ***JOBLOGSVR** The job log will be produced by a job log server. For more information about job log servers, refer to the Start Job Log Server (STRLOGSVR) command. ***PND** The job log will not be produced. The job log remains pending until removed. ***SYSVAL** The value is specified by the QLOGOUTPUT system value.

Table 117. JOB_DESCRIPTION_INFO view (continued)

Column Name	System Column Name	Data Type	Description
INQUIRY_MESSAGE_REPLY	INQ_REPLY	VARCHAR(8)	How inquiry messages are answered for jobs that use this job description. ***DFT** The system uses the default message reply to answer any inquiry messages issued while the job is running. The default reply is either defined in the message description or is the default system reply. ***RQD** The job requires an answer for any inquiry messages that occur while the job is running. ***SYSRPYL** The system reply list is checked to see if there is an entry for an inquiry message issued while the job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.
MESSAGE_LOGGING_LEVEL	LOG_LEVEL	SMALLINT	The type of information logged. **0** No messages are logged. **1** All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged. This includes the indication of job start, job end, and job completion status. **2** The following information is logged: • Level 1 information. • Request messages that result in a high-level message with a severity code greater than or equal to the logging severity cause the request message and all associated messages to be logged. **Note:** A high-level message is one that is sent to the program message queue of the program that receives the request message. For example, QCMD is an IBM-supplied request processing program that receives request messages. **3** The following information is logged: • Level 1 and 2 information. • All request messages. • Commands run by a CL program are logged if it is allowed by the logging of CL programs job attribute and the log attribute of the CL program. **4** The following information is logged: • All request messages and all messages with a severity greater than or equal to the message logging severity, including trace messages. • Commands run by a CL program are logged if it is allowed by the logging of CL programs job attribute and the log attribute of the CL program.
MESSAGE_LOGGING_SEVERITY	LOG_SEV	SMALLINT	The severity level that is used in conjunction with the logging level to determine which error messages are logged in the job log. The possible values are from 0 through 99.

Table 117. JOB_DESCRIPTION_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MESSAGE_LOGGING_TEXT	LOG_TEXT	VARCHAR(7)	The level of message text that is written in the job log when a message is logged according to the logging level and logging severity. ***MSG** Only the message text is written to the job log. ***NOLIST** If the job ends normally, no job log is produced. If the job ends abnormally (if the job end code is 20 or higher), a job log is produced. The messages that appear in the job log contain both the message text and the message help. ***SECLVL** Both the message text and the message help (cause and recovery) of the error message are written to the job log.
LOG_CL_PROGRAM_COMMANDS	LOG_CL	VARCHAR(4)	Whether or not commands are logged for CL programs that are run. ***NO** CL programs are not logged. ***YES** CL programs are logged.
DEVICE_RECOVERY_ACTION	DEVRECOVER	VARCHAR(13)	The action to take when an I/O error occurs for the interactive job's requesting program device. ***DSCENDRQS** Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level. ***DSCMSG** Disconnects the job when an I/O error occurs. When the job reconnects, the system sends a message to the application program indicating the job has reconnected and that the workstation device has recovered. ***ENDJOB** Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST). This message indicates that the job ended because of a device error. ***ENDJOBNOLIST** Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the history log (QHST). This message indicates that the job ended because of a device error. ***MSG** Signals the I/O error message to the application and lets the application program perform error recovery. ***SYSVAL** The value in the system value QDEVRCYACN at the time the job is started is used as the device recovery action for this job description.
TIME_SLICE_END_POOL	TIME_SLICE	VARCHAR(7)	Whether interactive jobs using this job description should be moved to another main storage pool when they reach time-slice end. ***BASE** The job is moved to the base pool when it reaches time-slice end. ***NONE** The job is not moved when it reaches time-slice end. ***SYSVAL** The system value is used.

Table 117. JOB_DESCRIPTION_INFO view (continued)

Column Name	System Column Name	Data Type	Description
ALLOW_MULTIPLE_THREADS	ALWMLTTHD	VARCHAR(4)	Whether or not the job is allowed to run with multiple user threads. This attribute does not prevent the operating system from creating system threads in the job. This attribute is not allowed to be changed once a job starts. This attribute applies to autostart jobs, prestart jobs, batch jobs submitted from job schedule entries, and jobs started by using the Submit Job (SBMJOB) and Batch Job (BCHJOB) commands. This attribute is ignored when starting all other types of jobs. This attribute should be set to *YES only in job descriptions that are used exclusively with functions that create multiple user threads. ***NO** The job is not allowed to run with multiple user threads. ***YES** The job is allowed to run with multiple user threads.
ASPGRP	ASPGRP	VARCHAR(10) Nullable	The name of the ASP group. This is the name of the primary ASP device in an ASP group or the name of an ASP device description. This specifies the initial ASP group setting for jobs using this job description. Contains the null value if jobs using this job description do not have an initial ASP group.
DDM_CONVERSATION	DDM_CONV	VARCHAR(5)	Whether the Distributed Data Management conversations are kept or dropped when they are not being used. The possible values are: ***DROP** The system ends a DDM-allocated conversation when there are no users. ***KEEP** The system keeps DDM conversation connections active when there are no users.

Examples

- Review information about the job queues associated with each job description.

```
SELECT JOB_DESCRIPTION_LIBRARY, JOB_DESCRIPTION,
       JOB_QUEUE_LIBRARY, JOB_QUEUE, JOB_QUEUE_PRIORITY
  FROM QSYS2.JOB_DESCRIPTION_INFO;
```

- Find the job descriptions that have APPLIB1 in their library list

```
SELECT JOB_DESCRIPTION_LIBRARY, JOB_DESCRIPTION, LIBRARY_LIST
  FROM QSYS2.JOB_DESCRIPTION_INFO
  WHERE LIBRARY_LIST LIKE '%APPLIB1%';
```

- Examine the library lists for every job description.

  Since the library list column returns a character string containing a list of libraries, to see the individual library names it needs to be broken apart. To do this, you can create a table function that takes the library list string and returns a list of library names.

```
CREATE OR REPLACE FUNCTION QGPL.GET_LIB_NAMES(JOBD_LIBL VARCHAR(2750),
                                              JOBD_LIBL_CNT INT)
  RETURNS TABLE(LIBL_POSITION INT, LIBRARY_NAME VARCHAR(10))
  BEGIN
    DECLARE IN_POS INT;
    DECLARE LIB_CNT INT;
    SET IN_POS = 1;
    SET LIB_CNT = 1;
    WHILE LIB_CNT <= JOBD_LIBL_CNT
      DO
      PIPE (LIB_CNT, RTRIM((SUBSTR(JOBD_LIBL, IN_POS, 10))));
      SET IN_POS = IN_POS + 11;
      SET LIB_CNT = LIB_CNT + 1;
    END WHILE;
    RETURN;
  END;
```

Now this function can be used to return the list of library names.

```
SELECT JOB_DESCRIPTION, JOB_DESCRIPTION_LIBRARY, LIBL_POSITION, LIBRARY_NAME
  FROM QSYS2.JOB_DESCRIPTION_INFO,
  TABLE (QGPL.GET_LIB_NAMES(LIBRARY_LIST, LIBRARY_LIST_COUNT)) X;
```

GET_JOB_INFO table function

The GET_JOB_INFO table function returns one row containing the information about a specific job.

▶▶─ GET_JOB_INFO ── (── *job-name* ──) ─▶◀

The schema is QSYS2.

To invoke this function, the caller must have *JOBCTL user special authority, or QIBM_DB_SQLADM or QIBM_DB_SYSMON function usage authority.

job-name
: A character or graphic string expression that identifies the name of a job. The special value of '*' indicates the current job.

The result of the function is a table containing a single row with the format shown in the following table. All the columns are nullable.

Table 118. GET_JOB_INFO table function

Column Name	Data Type	Description
V_JOB_STATUS	CHAR(10)	Status of the job. ***ACTIVE** Job is active. It could be a group job, system request job, or disconnected job. ***COMPLETE** Job is complete and job status information is no longer accessible. ***JOBQ** Job is currently on job queue. ***OUTQ** Job has completed running but has output on an output queue or the job log has not yet been written.
V_ACTIVE_JOB _STATUS	CHAR(4)	The active status of the initial thread of the job. For the list of values see Work Management API Attribute Descriptions in Application Programming Interfaces and search on "Active job status".
V_RUN_PRIORITY	INTEGER	The highest run priority allowed for any thread within this job.
V_SBS_NAME	CHAR(10)	Name of subsystem where job is running.
V_CPU_USED	BIGINT	The amount of CPU time (in milliseconds) that has been currently used by this job.
V_TEMP_STORAGE_USED_MB	INTEGER	The amount of auxiliary storage (in megabytes) that is currently allocated to this job.
V_AUX_IO_REQUESTED	BIGINT	The number of auxiliary I/O requests performed by the job across all routing steps. This includes both database and nondatabase paging.
V_PAGE_FAULTS	BIGINT	The number of times an active program referenced an address that was not in main storage during the current routing step of the specified job.
V_CLIENT_WRKSTNNAME	CHAR(255)	Value of the SQL CLIENT_WRKSTNNAME special register.
V_CLIENT_APPLNAME	CHAR(255)	Value of the SQL CLIENT_APPLNAME special register.
V_CLIENT_ACCTNG	CHAR(255)	Value of the SQL CLIENT_ACCTNG special register.
V_CLIENT_PROGRAMID	CHAR(255)	Value of the SQL CLIENT_PROGRAMID special register.
V_CLIENT_USERID	CHAR(255)	Value of the SQL CLIENT_USERID special register.
V_SQL_STATEMENT_TEXT	VARCHAR(10000)	Statement text of the last SQL statement to run or the SQL statement that is currently running.

Table 118. GET_JOB_INFO table function (continued)

Column Name	Data Type	Description
V_SQL_STMT_STATUS	CHAR(8)	The status of SQL within this job. **ACTIVE** An SQL statement is currently running **COMPLETE** At least one SQL statement has run and has completed Returns null if no SQL statement has been run.
V_SQL_STMT_START_TIMESTAMP	TIMESTAMP	The timestamp of the execution start for an active SQL statement. If there is no active SQL statement, the null value is returned.
V_QUERY_OPTIONS_LIB_NAME	CHAR(10)	The name of the QAQQINI options library in use for this job.
V_CLIENT_IP_ADDRESS	VARCHAR(45)	Client IP address being used by the job. Returns null when no client IP address exists or the job is using IPv6.
V_PJ_REUSE_COUNT	INTEGER	The number of times the prestart job has been used. The prestart job reuse count is incremented when a disconnect is processed for a prestart job. When the prestart job reuse count exceeds the prestart job maximum number of uses, the job is ended. Returns null if the job is not active or if the job is not a prestart job.
V_PJ_MAXUSE_COUNT	INTEGER	The maximum number of times the prestart job can be used before it is ended. A value of -1 is returned for *NOMAX. Returns null if the job is not active or if the job is not a prestart job.

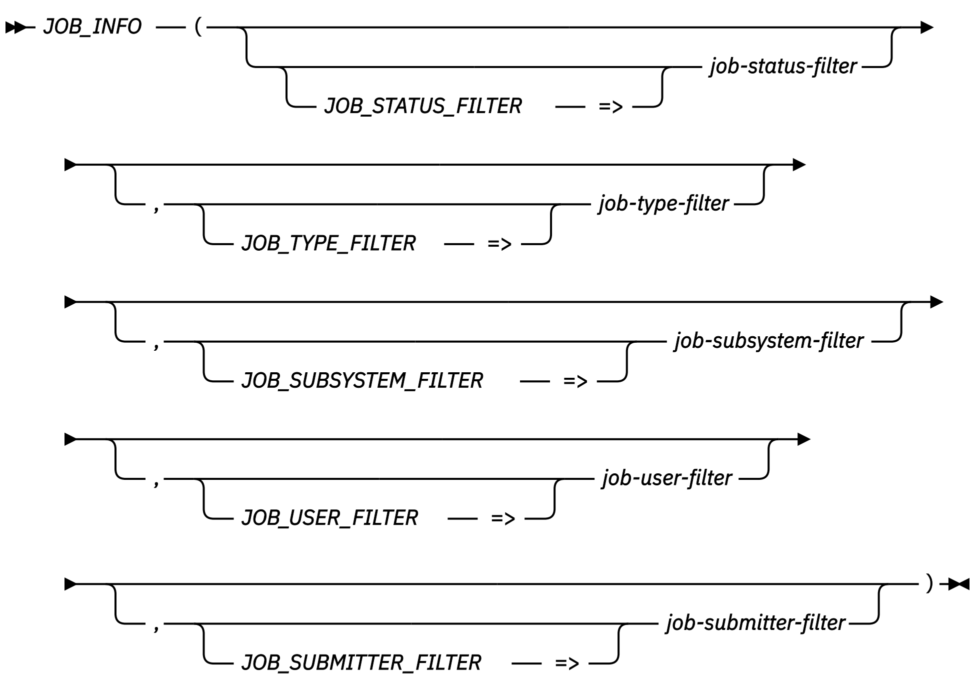
Example

Return information about job 347117/Quser/Qzdasoinit.

```
SELECT * FROM TABLE(QSYS2.GET_JOB_INFO('347117/Quser/Qzdasoinit')) A
```

JOB_INFO table function

The JOB_INFO table function returns one row for each job meeting the selection criteria. It returns information similar to what is returned by the Work with User Jobs (WRKUSRJOB), Work with Subsystem Jobs (WRKSBSJOB), and Work with Submitted Jobs (WRKSBMJOB) CL commands and the List Job (QUSLJOB) API.



The schema is QSYS2.

Authorization: None required.

job-status-filter
A character or graphic string expression that specifies the value to use as the job status filtering criteria. The string must be one of the following special values:

***ALL**
Jobs of any status including jobs on job queues, active jobs, and jobs on an output queue.

***ACTIVE**
Jobs that are active. You can use the QSYS2.ACTIVE_JOB_INFO table function to get additional details for these jobs.

***JOBQ**
Jobs that are not active because they are waiting on a job queue.

***OUTQ**
Jobs that have completed execution and have output on an output queue.

If this parameter is not provided, a value of ***ALL** is used.

job-type-filter
A character or graphic string expression that specifies the value to use as the job type filtering criteria. The string must be one of the following special values:

***ALL**
All types of user jobs, including interactive jobs and batch jobs.

***BATCH**
Only batch user jobs, including prestart jobs, batch immediate jobs, and autostart jobs.

***INTERACT**
Only interactive user jobs.

If this parameter is not provided, a value of ***ALL** is used.

job-subsystem-filter
A character or graphic string expression that specifies the subsystem value to use as the job subsystem filtering criteria. The string can be a subsystem name or the following value:

***ALL**
All jobs in all subsystems, including jobs that are on job queues and on output queues.

If a subsystem name is provided, only active jobs and jobs that are on job queues are found.

If this parameter is not provided, a value of ***ALL** is used.

job-user-filter
The USER special register or a character or graphic string expression that specifies the user profile name to use as the job user filtering criteria.

The string can be a user name or one of the following special values:

***ALL**
All jobs being processed under all user names.

***USER**
The user part of the qualified job name.

The USER special register is specified as a non-string value. It represents the current user of the job invoking the function.

If this parameter is not provided, the value of the USER special register is used.

job-submitter-filter
A character or graphic string expression that specifies the type of submitted jobs to return. The string must be one of the following values:

***ALL**
All submitted jobs.

***JOB**
Jobs that were submitted from the same job that is invoking this function.

***USER**
Jobs that were submitted from a job having the same user profile as the job invoking this function.

***WRKSTN**
Jobs that were submitted from the same work station as the job invoking this function.

If this parameter is not provided, a value of ***ALL** is used.

Restrictions:

- Only one of these filters can have a value other than *ALL: *job-subsystem-filter* and *job-submitter-filter*.
- If a value other than *ALL is specified for *job-submitter-filter*, you must specify *ALL for *job-user-filter*.

Notes:

- Jobs submitted with *NO specified for the Allow display by WRKSBMJOB (DSPSBMJOB) parameter of the SBMJOB command are not returned by this table function.

For each of the WRKSBMJOB, WRKSBSJOB, and WRKUSRJOB CL commands shown below, the corresponding invocation of JOB_INFO will return the same list of jobs. Note that to get exact equivalence, predicates must be added to some queries to achieve equivalent results:

- For equivalence with WRKUSRJOB, a query must always include the predicate WHERE JOB_TYPE NOT IN ('SBS','SYS','RDR','WTR')
- For equivalence with WRKSBSJOB SBS(*OUTQ) or WRKSBSJOB SBS(*ALL), a query must always include the predicate WHERE JOB_TYPE NOT IN ('SBS','SYS')

Table 119. Equivalent CL command and JOB_INFO invocations

CL Command	CL Parameters	JOB_INFO invocation

WRKSBMJOB	SBMFROM(*USER)	`SELECT * FROM TABLE(QSYS2.JOB_INFO(`
		`JOB_SUBMITTER_FILTER => '*USER',`
		`JOB_USER_FILTER => '*ALL'`
		`)) X`

	SBMFROM(*WRKSTN)	`SELECT * FROM TABLE(QSYS2.JOB_INFO(`
		`JOB_SUBMITTER_FILTER => '*WRKSTN',`
		`JOB_USER_FILTER => '*ALL'`
		`)) X`

	SBMFROM(*JOB)	`SELECT * FROM TABLE(QSYS2.JOB_INFO(`
		`JOB_SUBMITTER_FILTER => '*JOB',`
		`JOB_USER_FILTER => '*ALL'`
		`)) X`

Table 119. Equivalent CL command and JOB_INFO invocations (continued)

CL Command	CL Parameters	JOB_INFO invocation
WRKSBSJOB	SBS(QBATCH) USER(*ALL)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_SUBSYSTEM_FILTER => 'QBATCH', JOB_USER_FILTER => '*ALL')) X
	SBS(*JOBQ) USER(*ALL)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_STATUS_FILTER => '*JOBQ', JOB_USER_FILTER => '*ALL')) X
	SBS(*OUTQ) USER(JOEUSER)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_STATUS_FILTER => '*OUTQ', JOB_USER_FILTER => 'JOEUSER')) X WHERE JOB_TYPE NOT IN ('SBS','SYS')
	SBS(*ALL) USER(*ALL)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_STATUS_FILTER => '*ALL', JOB_USER_FILTER => '*ALL')) X WHERE JOB_TYPE NOT IN ('SBS','SYS')
WRKUSRJOB	USER(*) STATUS(*ALL) JOBTYPE(*ALL)	SELECT * FROM TABLE(QSYS2.JOB_INFO( )) X WHERE JOB_TYPE NOT IN ('SBS','SYS','RDR','WTR')
	USER(*) STATUS(*ALL) JOBTYPE(*INTERAC T)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_TYPE_FILTER => '*INTERACT')) X WHERE JOB_TYPE NOT IN ('SBS','SYS','RDR','WTR')
	USER(JOEUSER) STATUS(*ACTIVE) JOBTYPE(*ALL)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_USER_FILTER => 'JOEUSER', JOB_STATUS_FILTER => '*ACTIVE')) X WHERE JOB_TYPE NOT IN ('SBS','SYS','RDR','WTR')
	USER(*) STATUS(*OUTQ) JOBTYPE(*ALL)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_STATUS_FILTER => '*OUTQ')) X WHERE JOB_TYPE NOT IN ('SBS','SYS','RDR','WTR')
	USER(*ALL) STATUS(*JOBQ) JOBTYPE(*BATCH)	SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_USER_FILTER => '*ALL', JOB_STATUS_FILTER => *JOBQ', JOB_TYPE_FILTER =>'*BATCH')) X WHERE JOB_TYPE NOT IN ('SBS','SYS','RDR','WTR')

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 120. JOB_INFO table function

Column Name	Data Type	Description
JOB_NAME	VARCHAR(28)	The qualified job name.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
JOB_INFORMATION	VARCHAR(12)	Indicates whether information is available for the job. **NO** The information is not available because the job was not accessible. **YES** The information is available. When this value is NO, all columns other than JOB_NAME return the null value.
JOB_STATUS	VARCHAR(6)	The status of the job. **ACTIVE** The job has started, and it can use system resources (processing unit, main storage, and so on). This does not guarantee that the job is currently running, however. For example, an active job may be in one of the following states where it is not in a position to use system resources: • The Hold Job (HLDJOB) command holds the job; the Release Job (RLSJOB) command allows the job to run again. • The Transfer Group Job (TFRGRPJOB) or Transfer Secondary Job (TFRSECJOB) command suspends the job. When control returns to the job, the job can run again. • The job is disconnected using the Disconnect Job (DSCJOB) command. When the interactive user signs back on, thereby connecting back into the job, the job can run again. • The job is waiting for any reason. For example, when the job receives the reply for an inquiry message, the job can start running again. **JOBQ** The job is currently on a job queue. The job possibly was previously active and was placed back on the job queue because of the Transfer Job (TFRJOB) or Transfer Batch Job (TFRBCHJOB) command, or the job was never active because it was just submitted. **OUTQ** The job has completed running and has spooled output that has not yet printed or the job's job log has not yet been written.
JOB_TYPE	VARCHAR(3)	The type of job. **ASJ** Autostart **BCH** Batch **BCI** Batch Immediate **EVK** Started by a procedure start request **INT** Interactive **M36** Advanced 36 server job **MRT** Multiple requester terminal **PDJ** Print driver job **PJ** Prestart job **RDR** Spool reader **SBS** Subsystem monitor **SYS** System **WTR** Spool writer

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
JOB_TYPE_ENHANCED	VARCHAR(28)	The combined job type and job subtype values. **ALTERNATE_SPOOL_USER** Batch - alternate spool user **AUTOSTART** Autostart job **BATCH** Batch job **BATCH_IMMEDIATE** Batch immediate job **BATCH_MRT** Batch - System/36 multiple requester terminal (MRT) job **COMM_PROCEDURE_START_REQUEST** Communications job - procedure start request job **INTERACTIVE** Interactive job **INTERACTIVE_GROUP** Interactive job - Part of group **INTERACTIVE_SYSREQ** Interactive job - Part of system request pair **INTERACTIVE_SYSREQ_AND_GROUP** Interactive job - Part of system request pair and part of a group **PRESTART** Prestart job **PRESTART_BATCH** Prestart batch job **PRESTART_COMM** Prestart communications job **READER** Reader job **SUBSYSTEM** Subsystem job **SYSTEM** System job (all system jobs including SCPF) **WRITER** Writer job (including both spool writers and print drivers)
JOB_SUBSYSTEM	VARCHAR(10)	The name of the subsystem for the job. Contains the null value if the job has no subsystem.
JOB_DATE	VARCHAR(10)	The date that is assigned to the job, in *ISO format. The job date remains the same for the duration of the job unless it is changed by the user. Can also contain the following special value: **SYSVAL** This job will use the system date. Contains the null value if JOB_STATUS is OUTQ.
JOB_DESCRIPTION_LIBRARY	VARCHAR(10)	The name of the library containing the job description. Contains the null value if JOB_DESCRIPTION is null.
JOB_DESCRIPTION	VARCHAR(10)	The name of the job description used for this job. Contains the null value if the job has no job description.
JOB_ACCOUNTING_CODE	VARCHAR(15)	An identifier assigned to the job by the system to collect resource use information for the job when job accounting is active. Contains the null value if the job has no accounting code.
SUBMITTER_JOB_NAME	VARCHAR(28)	The qualified job name of the submitter's job. Contains the null value if the job has no submitter.
SUBMITTER_MESSAGE_QUEUE_LIBRARY	VARCHAR(10)	The name of the library containing the message queue. Contains the null value if the job has no submitter.
SUBMITTER_MESSAGE_QUEUE	VARCHAR(10)	The name of the message queue where the system sends a completion message when a batch job ends. Contains the null value if the job has no submitter.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
SERVER_TYPE	VARCHAR(30)	The type of server represented by the job. See Server table for a list of server type values. Contains the null value if the job is not part of a server.
JOB_ENTERED_SYSTEM_TIME	TIMESTAMP(0)	The timestamp for when the job was placed on the system.
JOB_SCHEDULED_TIME	TIMESTAMP(0)	The timestamp for when the job is scheduled to become active. Contains the null value if this is not a scheduled job.
JOB_ACTIVE_TIME	TIMESTAMP(0)	The time the job began to run on the system. Contains the null value if the job did not become active.
JOB_END_TIME	TIMESTAMP(0)	The timestamp for when the job completed running on the system. Contains the null value if the job has not ended.
JOB_END_SEVERITY	SMALLINT	The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message, whose severity is equal to or greater than this value, to the request processing program.
COMPLETION_STATUS	VARCHAR(8)	The completion status of the job. **ABNORMAL** The job completed abnormally. **NORMAL** The job completed normally. Contains the null value if this the job has not completed.
JOB_END_REASON	VARCHAR(60)	The most recent action that caused the job to end. Contains one of the following values: • JOB ENDED DUE TO A DEVICE ERROR • JOB ENDED DUE TO A SIGNAL • JOB ENDED DUE TO AN UNHANDLED ERROR • JOB ENDED DUE TO THE CPU LIMIT BEING EXCEEDED • JOB ENDED DUE TO THE DISCONNECT TIME INTERVAL BEING EXCEEDED • JOB ENDED DUE TO THE INACTIVITY TIME INTERVAL BEING EXCEEDED • JOB ENDED DUE TO THE MESSAGE SEVERITY LEVEL BEING EXCEEDED • JOB ENDED DUE TO THE STORAGE LIMIT BEING EXCEEDED • JOB ENDED WHILE IT WAS STILL ON A JOB QUEUE • JOB ENDING ABNORMALLY • JOB ENDING IMMEDIATELY • JOB ENDING IN NORMAL MANNER • JOB ENDING NORMALLY AFTER A CONTROLLED END WAS REQUESTED • SYSTEM ENDED ABNORMALLY Contains the null value if job is not currently ending.
JOB_QUEUE_LIBRARY	VARCHAR(10)	The name of the library containing the job queue. Contains the null value if JOB_STATUS is OUTQ or if job is not on a job queue and the job is not a batch job that was started from a job queue.
JOB_QUEUE_NAME	VARCHAR(10)	The name of the job queue that the job is currently on, or that the job was on if it is currently active. Contains the null value if JOB_STATUS is OUTQ or if job is not on a job queue and the job is not a batch job that was started from a job queue.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
JOB_QUEUE_STATUS	VARCHAR(9)	The status of this job on the job queue. **HELD** This job is being held on the job queue. **RELEASED** This job is ready to be selected. **SCHEDULED** This job will run as scheduled. Contains the null value if the job is not on a job queue.
JOB_QUEUE_PRIORITY	SMALLINT	The scheduling priority of the job compared to other jobs on the same job queue. The highest priority is 0 and the lowest is 9. Contains the null value if JOB_STATUS is not JOBQ.
JOB_QUEUE_TIME	TIMESTAMP(0)	The timestamp when the job was put on the job queue. Contains the null value if this the job is not on a job queue.
JOB_MESSAGE_QUEUE_MAXIMUM_SIZE	SMALLINT	The maximum size, in megabytes, that the job message queue can become. The range is 2 to 64. Contains the null value if JOB_QUEUE_NAME is null.
JOB_MESSAGE_QUEUE_FULL_ACTION	VARCHAR(8)	The action to take when the message queue is full. ***NOWRAP** When the job message queue is full, do not wrap. This action causes the job to end. ***PRTWRAP** When the job message queue is full, wrap the message queue and print the messages that are being overlaid because of the wrapping. ***WRAP** When the job message queue is full, wrap to the beginning and start filling again. Contains the null value if JOB_QUEUE_NAME is null.
ALLOW_MULTIPLE_THREADS	VARCHAR(3)	Indicates whether this job allows multiple user threads. This attribute does not prevent the operating system from creating system threads in the job. **NO** This job does not allow multiple user threads. **YES** This job allows multiple user threads.
PEAK_TEMPORARY_STORAGE	INTEGER	The maximum amount of auxiliary storage, in megabytes, that the job has used. Contains the null value if JOB_STATUS is OUTQ or for a job on a job queue if a value has not been set for the job.
DEFAULT_WAIT	INTEGER	The default maximum time, in seconds, that a thread in the job waits for a system instruction, such as a LOCK machine interface (MI) instruction, to acquire a resource. Contains the null value if there is no maximum, if JOB_STATUS is OUTQ, or for a job on a job queue if a value has not been set for the job.
MAXIMUM_PROCESSING_TIME_ ALLOWED	INTEGER	The maximum processing unit time, in milliseconds, that the job can use. If the job consists of multiple routing steps, this is the maximum processing unit time that the current routing step can use. If the maximum time is exceeded, the job is held. Contains the null value if JOB_STATUS is OUTQ or if no maximum amount of processing unit time has been defined.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
MAXIMUM_TEMPORARY_STORAGE_ ALLOWED	INTEGER	The maximum amount of auxiliary storage, in megabytes, that the job can use. If the job consists of multiple routing steps, this is the maximum temporary storage that the routing step can use. This temporary storage is used for storage required by the program itself and by implicitly created internal system objects used to support the routing step. (It does not include storage for objects in the QTEMP library.) If the maximum temporary storage is exceeded, the job is held. This does not apply to the use of permanent storage, which is controlled through the user profile. Contains the null value if JOB_STATUS is OUTQ or if no maximum amount of temporary storage has been defined.
TIME_SLICE	INTEGER	The maximum amount of processor time, in milliseconds, given to each thread in this job before other threads in this job and in other jobs are given the opportunity to run. The time slice establishes the amount of time needed by a thread in this job to accomplish a meaningful amount of processing. At the end of the time slice, the thread might be put in an inactive state so that other threads can become active in the storage pool. Values range from 8 through 9999999. Contains the null value if JOB_STATUS is OUTQ or for a job on a job queue if a value has not been set for the job.
JOB_SWITCHES	CHAR(8)	The current setting of the job switches used by this job. Contains the null value no job switches are set.
ROUTING_DATA	VARCHAR(80)	The routing data that is used to determine the routing entry that identifies the program to start for the routing step. Contains the null value if there is no routing data for this job.
CCSID	INTEGER	The coded character set identifier (CCSID) used for this job. Contains the null value if no CCSID is defined for this job.
CHARACTER_IDENTIFIER_CONTROL	VARCHAR(9)	The character identifier control for the job. This attribute controls the type of CCSID conversion that occurs for display files, printer files, and panel groups. The *CHRIDCTL special value must be specified on the CHRID command parameter on the create, change, or override command for display files, printer files, and panel groups before this attribute will be used. ***DEVD** The *DEVD special value performs the same function as on the CHRID command parameter for display files, printer files, and panel groups. ***JOBCCSID** The *JOBCCSID special value performs the same function as on the CHRID command parameter for display files, printer files, and panel groups.
SORT_SEQUENCE_LIBRARY	VARCHAR(10)	The name or the library that contains the sort sequence table. Contains the null value if no sort sequence table is defined for this job or if SORT_SEQUENCE_NAME is a special value.
SORT_SEQUENCE_NAME	VARCHAR(10)	The name of the sort sequence table associated with this job. Contains the null value if no sort sequence table is defined for this job.
LANGUAGE_ID	CHAR(3)	The language identifier associated with this job.
COUNTRY_ID	CHAR(2)	The country or region identifier associated with this job.
DATE_FORMAT	CHAR(4)	The date format used for this job. ***DMY** Day, month, year format. ***JUL** Julian format (year and day). ***MDY** Month, day, year format. ***YMD** Year, month, day format.
DATE_SEPARATOR	CHAR(1)	The date separator used for this job.
TIME_SEPARATOR	CHAR(1)	The time separator used for this job.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
DECIMAL_FORMAT	VARCHAR(6)	The decimal format used for this job. ***BLANK** Uses a period for a decimal point, a comma for a 3-digit grouping character, and zero-suppress to the left of the decimal point. **J** Uses a comma for a decimal point and a period for a 3-digit grouping character. The zero-suppression character is in the second position (rather than the first) to the left of the decimal notation. Balances with zero values to the left of the comma are written with one leading zero (0,04). The J entry also overrides any edit codes that might suppress the leading zero. **I** Uses a comma for a decimal point, a period for a 3-digit grouping character, and zero-suppress to the left of the decimal point.
TIME_ZONE_DESCRIPTION_NAME	VARCHAR(10)	The name of the time zone description that is used to calculate local job time.
MESSAGE_LOGGING_LEVEL	SMALLINT	The type of information that is logged. **0** No messages are logged. **1** All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged. This includes the indication of job start, job end and job completion status. **2** The following information is logged: • Level 1 information • Request messages that result in a high-level message with a severity code greater than or equal to the logging severity cause the request message and all associated messages to be logged. Note: A high-level message is one that is sent to the program message queue of the program that receives the request message. For example, QCMD is an IBM-supplied request processing program that receives request messages. **3** The following information is logged: • Level 1 and 2 information • All request messages • Commands run by a CL program are logged if it is allowed by the logging of CL programs job attribute and the log attribute of the CL program. **4** The following information is logged: • All request messages and all messages with a severity greater than or equal to the message logging severity, including trace messages. • Commands run by a CL program are logged if it is allowed by the logging of CL programs job attribute and the log attribute of the CL program.
MESSAGE_LOGGING_SEVERITY	SMALLINT	The severity level that is used in conjunction with the logging level to determine which error messages are logged in the job log. The values range from 0 through 99.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
MESSAGE_LOGGING_TEXT	VARCHAR(7)	The level of message text that is written in the job log when a message is logged according to the logging level and logging severity. ***MSG** Only the message text is written to the job log. ***NOLIST** If the job ends normally, no job log is produced. If the job ends abnormally (the job end code is 20 or higher), a job log is produced. The messages that appear in the job log contain both the message text and the message help. ***SECLVL** Both the message text and the message help (cause and recovery) of the error message are written to the job log.
LOG_CL_PROGRAM_COMMANDS	VARCHAR(4)	Specifies whether or not commands are logged for CL programs that are run. ***NO** Commands are not logged. ***YES** Commands are logged.
STATUS_MESSAGE	VARCHAR(7)	Specifies whether status messages are displayed for this job. ***NONE** This job does not display status messages. ***NORMAL** This job displays status messages.
INQUIRY_MESSAGE_REPLY	VARCHAR(8)	Specifies how the job answers inquiry messages. ***RQD** The job requires an answer for any inquiry messages that occur while this job is running. ***DFT** The system uses the default message reply to answer any inquiry messages issued while this job is running. The default reply is either defined in the message description or is the default system reply. ***SYSRPYL** The system reply list is checked to see if there is an entry for an inquiry message issued while this job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.
BREAK_MESSAGE	VARCHAR(7)	Specifies how this job handles break messages. ***HOLD** The message queue holds break messages until a user or program requests them. The work station user uses the Display Message (DSPMSG) command to display the messages; a program must issue a Receive Message (RCVMSG) command to receive a message and handle it. ***NORMAL** The message queue status determines break message handling. ***NOTIFY** The system notifies the job's message queue when a message arrives. For interactive jobs, the audible alarm sounds if there is one, and the message-waiting light comes on.
JOB_LOG_OUTPUT	VARCHAR(10)	Specifies how the job log will be produced when the job completes. ***JOBEND** The job log will be produced by the job itself. If the job cannot produce its own job log, the job log will be produced by a job log server. For example, a job does not produce its own job log when the system is processing a Power Down System (PWRDWNSYS) command. ***JOBLOGSVR** The job log will be produced by a job log server. For more information about job log servers, refer to the Start Job Log Server (STRLOGSVR) command. ***PND** The job log will not be produced. The job log remains pending until removed.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
JOB_LOG_PENDING	VARCHAR(3)	Specifies whether there is a job log that has not yet been written. The writing of the job log may become pending based on the value of the job log output job attribute when the job completes its activity. **NO** Job log is not pending. **YES** Job log is pending.
OUTPUT_QUEUE_PRIORITY	SMALLINT	The output priority for spooled output files that this job produces. The highest priority is 0, and the lowest is 9.
OUTPUT_QUEUE_LIBRARY	VARCHAR(10)	The name of the library that contains the default output queue.
OUTPUT_QUEUE_NAME	VARCHAR(10)	The name of the default output queue that is used for spooled output produced by this job and the name of the library that contains the output queue. The default output queue is only for spooled printer files that specify *JOB for the output queue.
SPOOLED_FILE_ACTION	VARCHAR(7)	Specifies whether spooled files are accessed through job interfaces after the job has completed is normal activity. ***DETACH** The spooled files are detached from the job when the job completes its activity. ***KEEP** When the job completes its activity, as long as at least one spooled file for the job exists in the system auxiliary storage pool (ASP 1) or in a basic user ASP (ASPs 2-32), the spooled files are kept with the job and the status of the job is updated to indicate that the job has completed. If all remaining spooled files for the job are in independent ASPs (ASPs 33-255), the spooled files will be detached from the job and the job will be removed from the system.
PRINTER_DEVICE_NAME	VARCHAR(10)	The printer device used for printing output from this job.
PRINT_KEY_FORMAT	VARCHAR(7)	Specifies whether border and header information is provided when the Print key is pressed. ***NONE** The border and header information is not included with output from the Print key. ***PRTBDR** The border information is included with output from the Print key. ***PRTHDR** The header information is included with output from the Print key. ***PRTALL** The border and header information is included with output from the Print key.
PRINT_TEXT	VARCHAR(30)	The line of text that is printed at the bottom of each page of printed output for the job. Contains the null value if there is no text defined to print at the bottom of each page.
DEVICE_NAME	VARCHAR(10)	The name of the device as identified to the system. For an interactive job it is the device where the job started. Contains the null value if this is not an interactive job.

Table 120. JOB_INFO table function (continued)

Column Name	Data Type	Description
DEVICE_RECOVERY_ACTION	VARCHAR(13)	The action taken for interactive jobs when an I/O error occurs for the job's requesting program device. ***DSCENDRQS** Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level. ***DSCMSG** Disconnects the job when an I/O error occurs. When the job reconnects, the system sends an error message to the application program, indicating the job has reconnected and that the work station device has recovered. ***ENDJOB** Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST) indicating the job ended because of a device error. ***ENDJOBNOLIST** Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the QHST log indicating the job ended because of a device error. ***MSG** Signals the I/O error message to the application and lets the application program perform error recovery. Contains the null value if this is not an interactive job.
DDM_CONVERSATION	VARCHAR(5)	Specifies whether connections using distributed data management (DDM) protocols remain active when they are not being used. The connections include APPC conversations, active TCP/IP connections or Opti-Connect connections. ***DROP** The system ends a DDM connection when there are no users. Examples include when an application closes a DDM file, or when a DRDA application runs a SQL DISCONNECT statement. ***KEEP** The system keeps DDM connections active when there are no users, except for the following: • The routing step ends on the source system. The routing step ends when the job ends or when the job is rerouted to another routing step. • The Reclaim Distributed Data Management Conversation (RCLDDMCNV) command or the Reclaim Resources (RCLRSC) command runs. • A communications failure or an internal failure occurs. • A DRDA connection to an application server not running on the system ends.
MODE_NAME	VARCHAR(8)	The mode name of the advanced program-to-program communications device that started the job. The following special value may be returned: ***BLANK** The mode name is a blank name. Contains the null value if the job is not using advanced program-to-program communications (APPC).
UNIT_OF_WORK_ID	CHAR(24)	The unit of work ID is used to track jobs across multiple systems. Contains the null value if the job is not associated with a source or target system using advanced program-to-program communications (APPC).
INTERNAL_JOB_ID	BINARY(16)	The internal job identifier.

Examples

- Find all interactive jobs.

```
SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_TYPE_FILTER => '*INTERACT')) X;
```

- Find jobs submitted by SCOTTF that have not been started.

```
SELECT * FROM TABLE(QSYS2.JOB_INFO(JOB_USER_FILTER   => 'SCOTTF',
                                   JOB_STATUS_FILTER => '*JOBQ')) X;
```

JOB_QUEUE_INFO view

The JOB_QUEUE_INFO view returns one row for each job queue.

The values returned for the columns in the view are similar to the values returned by the Work with Job Queue (WRKJOBQ) CL command and the Retrieve Job Queue Information (QSPRJOBQ) API.

Authorization: Rows will be returned for job queues when the caller has:

- Execute authority to the job queue library and
  - Read authority to the job queue, or
  - *JOBCTL special authority and the job queue has OPRCTL(*YES), or
  - *SPLCTL special authority

The following table describes the columns in the view. The system name is JOBQ_INFO. The schema is QSYS2.

Table 121. JOB_QUEUE_INFO view

Column Name	System Column Name	Data Type	Description
JOB_QUEUE_NAME	JOBQ	VARCHAR(10)	The name of the job queue.
JOB_QUEUE_LIBRARY	JOBQ_LIB	VARCHAR(10)	The name of the library that contains the job queue.
JOB_QUEUE_STATUS	STATUS	VARCHAR(8)	The status of the job queue. **HELD** The queue is held. **RELEASED** The queue is released.
NUMBER_OF_JOBS	JOBS	INTEGER	The number of jobs in the queue.
SUBSYSTEM_NAME	SUB_NAME	VARCHAR(10) Nullable	The name of the subsystem that can receive jobs from this job queue. Contains the null value if this job queue is not associated with an active subsystem.
SUBSYSTEM_LIBRARY_NAME	SUBLIB_NAM	VARCHAR(10) Nullable	The library in which the subsystem description resides. Contains the null value if this job queue is not associated with an active subsystem.
SEQUENCE_NUMBER	SEQNO	INTEGER Nullable	The job queue entry sequence number. The subsystem uses this number to determine the order in which job queues are processed. Jobs from the queue with the lowest sequence number are processed first. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS	MAX_JOBS	INTEGER Nullable	The maximum number of jobs that can be active at the same time through this job queue entry. A value of -1 indicates *NOMAX, no maximum number of jobs is defined. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS	ACT_JOBS	INTEGER Nullable	The current number of jobs that are active that came through this job queue entry. Contains the null value if this job queue is not associated with an active subsystem.
HELD_JOBS	HELD_JOBS	INTEGER	The current number of jobs that are in *HELD status. This is the sum of the 10 HELD_JOBS_PRIORITY_n columns.

Table 121. JOB_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
RELEASED_JOBS	RLS_JOBS	INTEGER	The current number of jobs that are in *RELEASED status. This is the sum of the 10 RELEASED_JOBS_PRIORITY_n columns.
SCHEDULED_JOBS	SCHED_JOBS	INTEGER	The current number of jobs that are in *SCHEDULED status. This is the sum of the 10 SCHEDULED_JOBS_PRIORITY_n columns.
TEXT_DESCRIPTION	TEXT	VARCHAR(50) Nullable	Text that describes the job queue. Contains the null value if there is no text description for the job queue.
OPERATOR_CONTROLLED	OPR_CTRL	VARCHAR(4)	Whether users with job control authority are allowed to control this job queue and manage the jobs on the queue. Users have job control authority if SPCAUT(*JOBCTL) is specified in their user profile. ***NO** This queue and its jobs cannot be controlled by users with job control authority unless they also have other special authority. ***YES** Users with job control authority can control the queue and manage the jobs on the queue.
AUTHORITY_TO_CHECK	ALL_AUTH	VARCHAR(7)	Whether the user must be the owner of the queue in order to control the queue by holding or releasing the queue. ***DTAAUT** Any user with *READ, *ADD, or *DELETE authority to the job queue can control the queue. ***OWNER** Only the owner of the job queue can control the queue.
MAXIMUM_ACTIVE_JOBS_PRIORITY_1	MAXIMUM1	INTEGER Nullable	The maximum number of priority 1 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_PRIORITY_2	MAXIMUM2	INTEGER Nullable	The maximum number of priority 2 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_PRIORITY_3	MAXIMUM3	INTEGER Nullable	The maximum number of priority 3 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_PRIORITY_4	MAXIMUM4	INTEGER Nullable	The maximum number of priority 4 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_PRIORITY_5	MAXIMUM5	INTEGER Nullable	The maximum number of priority 5 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_PRIORITY_6	MAXIMUM6	INTEGER Nullable	The maximum number of priority 6 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.

Table 121. JOB_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MAXIMUM_ACTIVE_JOBS_ PRIORITY_7	MAXIMUM7	INTEGER Nullable	The maximum number of priority 7 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_ PRIORITY_8	MAXIMUM8	INTEGER Nullable	The maximum number of priority 8 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
MAXIMUM_ACTIVE_JOBS_ PRIORITY_9	MAXIMUM9	INTEGER Nullable	The maximum number of priority 9 jobs that can be active at the same time. A value of -1 indicates *NOMAX, no maximum number of jobs. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_0	ACTIVE0	INTEGER Nullable	The number of priority 0 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_1	ACTIVE1	INTEGER Nullable	The number of priority 1 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_2	ACTIVE2	INTEGER Nullable	The number of priority 2 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_3	ACTIVE3	INTEGER Nullable	The number of priority 3 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_4	ACTIVE4	INTEGER Nullable	The number of priority 4 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_5	ACTIVE5	INTEGER Nullable	The number of priority 5 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_6	ACTIVE6	INTEGER Nullable	The number of priority 6 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_7	ACTIVE7	INTEGER Nullable	The number of priority 7 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_8	ACTIVE8	INTEGER Nullable	The number of priority 8 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
ACTIVE_JOBS_PRIORITY_9	ACTIVE9	INTEGER Nullable	The number of priority 9 jobs that are active. Contains the null value if this job queue is not associated with an active subsystem.
RELEASED_JOBS_PRIORITY_0	RELEASED0	INTEGER	The number of priority 0 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_1	RELEASED1	INTEGER	The number of priority 1 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_2	RELEASED2	INTEGER	The number of priority 2 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_3	RELEASED3	INTEGER	The number of priority 3 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_4	RELEASED4	INTEGER	The number of priority 4 jobs currently sitting on the job queue in *RELEASED status.

Table 121. JOB_QUEUE_INFO view (continued)

Column Name	System Column Name	Data Type	Description
RELEASED_JOBS_PRIORITY_5	RELEASED5	INTEGER	The number of priority 5 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_6	RELEASED6	INTEGER	The number of priority 6 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_7	RELEASED7	INTEGER	The number of priority 7 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_8	RELEASED8	INTEGER	The number of priority 8 jobs currently sitting on the job queue in *RELEASED status.
RELEASED_JOBS_PRIORITY_9	RELEASED9	INTEGER	The number of priority 9 jobs currently sitting on the job queue in *RELEASED status.
SCHEDULED_JOBS_PRIORITY_0	SCHEDULED0	INTEGER	The number of priority 0 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_1	SCHEDULED1	INTEGER	The number of priority 1 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_2	SCHEDULED2	INTEGER	The number of priority 2 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_3	SCHEDULED3	INTEGER	The number of priority 3 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_4	SCHEDULED4	INTEGER	The number of priority 4 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_5	SCHEDULED5	INTEGER	The number of priority 5 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_6	SCHEDULED6	INTEGER	The number of priority 6 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_7	SCHEDULED7	INTEGER	The number of priority 7 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_8	SCHEDULED8	INTEGER	The number of priority 8 jobs currently sitting on the job queue in *SCHEDULED status.
SCHEDULED_JOBS_PRIORITY_9	SCHEDULED9	INTEGER	The number of priority 9 jobs currently sitting on the job queue in *SCHEDULED status.
HELD_JOBS_PRIORITY_0	HELD0	INTEGER	The number of priority 0 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_1	HELD1	INTEGER	The number of priority 1 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_2	HELD2	INTEGER	The number of priority 2 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_3	HELD3	INTEGER	The number of priority 3 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_4	HELD4	INTEGER	The number of priority 4 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_5	HELD5	INTEGER	The number of priority 5 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_6	HELD6	INTEGER	The number of priority 6 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_7	HELD7	INTEGER	The number of priority 7 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_8	HELD8	INTEGER	The number of priority 8 jobs currently sitting on the job queue in *HELD status.
HELD_JOBS_PRIORITY_9	HELD9	INTEGER	The number of priority 9 jobs currently sitting on the job queue in *HELD status.

Example

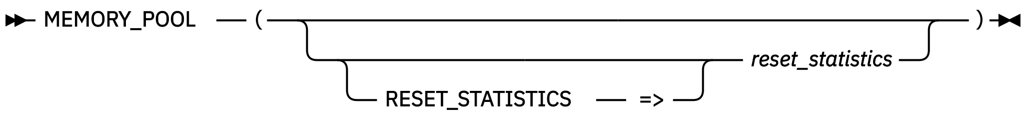
- Examine the job queues with the largest number of active jobs

```
SELECT * FROM QSYS2.JOB_QUEUE_INFO
  WHERE ACTIVE_JOBS IS NOT NULL
  ORDER BY NUMBER_OF_JOBS DESC;
```

MEMORY_POOL table function

The MEMORY_POOL table function returns one row for every pool.

The information returned is similar to the detail seen from the Work System Status (WRKSYSSTS) command.



The schema is QSYS2.

reset_statistics

A character or graphic string expression that contains a value of YES or NO.

If this parameter has a value of YES, statistics are reset such that the time of this query execution is used as the new baseline. The columns that contain this statistical data have names that are prefixed with ELAPSED_. Future invocations of MEMORY_POOL within this connection will return statistical detail relative to the new baseline. If this parameter has a value of NO, statistics are not reset for the invocation. If this parameter is not specified, the default is NO.

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 122. MEMORY_POOL table function

Column Name	Data Type	Description
SYSTEM_POOL_ID	INTEGER	The system-related pool identifier for each of the system storage pools that currently has main storage allocated to it.
POOL_NAME	VARCHAR(10)	The name of this storage pool. The name may be a number, in which case it is a private pool associated with a subsystem, or one of the following special values. ***MACHINE** The specified pool definition is defined to be the machine pool. ***BASE** The specified pool definition is defined to be the base system pool, which can be shared with other subsystems. ***INTERACT** The specified pool definition is defined to be the shared pool used for the QINTER subsystem. ***SPOOL** The specified pool definition is defined to be the shared pool used for spooled writers. ***SHRPOOL*x*** The specified pool definition is defined to be a shared pool.
CURRENT_SIZE	DECIMAL(20,2)	The amount of main storage, in megabytes, in the pool.
RESERVED_SIZE	DECIMAL(10,2)	The amount of storage, in megabytes, in the pool reserved for system use (for example, for save/restore operations).
DEFINED_SIZE	DECIMAL(20,2)	The size of the pool, in megabytes, as defined in the shared pool, subsystem description, or system value QMCHPOOL. Contains the null value for a pool without a defined size.
MAXIMUM_ACTIVE_THREADS	INTEGER	The maximum number of threads that can be active in the pool at any one time.
CURRENT_THREADS	INTEGER	The number of threads currently using the pool.
CURRENT_INELIGIBLE_THREADS	INTEGER	The number of ineligible threads in the pool.

Table 122. MEMORY_POOL table function (continued)

Column Name	Data Type	Description
STATUS	VARCHAR(8)	The status of the pool. **ACTIVE** Pool is currently active. **INACTIVE** Pool is currently not active.
SUBSYSTEM_LIBRARY_NAME	VARCHAR(10)	The library containing the subsystem name. Contains the null value for shared pools.
SUBSYSTEM_NAME	VARCHAR(10)	The subsystem with which this storage pool is associated. Contains the null value for shared pools.
DESCRIPTION	VARCHAR(50)	The description of the shared pool. Contains the null value for private pools or if a description does not exist for a shared pool.
PAGING_OPTION	VARCHAR(10)	Whether the system will dynamically adjust the paging characteristics of the storage pool for optimum performance. ***FIXED** The system does not dynamically adjust the paging characteristics. ***CALC** The system dynamically adjusts the paging characteristics. **USRDFN** The system does not dynamically adjust the paging characteristics for the storage pool but uses values that have been defined through the QWCCHGTN API.
ELAPSED_TIME	INTEGER	The time, in seconds, since the measurement start time.
ELAPSED_DATABASE_FAULTS	DECIMAL(10,1)	The rate, in page faults per second, of database page faults against pages containing either database access paths or data.
ELAPSED_NON_DATABASE_FAULTS	DECIMAL(10,1)	The rate, in page faults per second, of nondatabase page faults against pages other than those designated as database pages.
ELAPSED_TOTAL_FAULTS	DECIMAL(10,1)	The rate, in page faults per second, of database faults and non-database faults.
ELAPSED_DATABASE_PAGES	DECIMAL(10,1)	The rate, in pages per second, at which database pages are brought into the storage pool.
ELAPSED_NON_DATABASE_PAGES	DECIMAL(10,1)	The rate in pages per second at which nondatabase pages are brought into the storage pool.
ELAPSED_ACTIVE_TO_WAIT	DECIMAL(10,1)	The rate, in transitions per minute, of transitions of threads from an active condition to a waiting condition.
ELAPSED_WAIT_TO_INELIGIBLE	DECIMAL(10,1)	The rate, in transitions per minute, of transitions of threads from a waiting condition to an ineligible condition.
ELAPSED_ACTIVE_TO_INELIGIBLE	DECIMAL(10,1)	The rate, in transitions per minute, of transitions of threads from an active condition to an ineligible condition.
TUNING_PRIORITY	INTEGER	The priority of the shared storage pool used by the system when making automatic performance adjustments. Contains the null value for private pools defined in subsystem descriptions.
TUNING_MINIMUM_SIZE	DECIMAL(10,2)	The minimum amount of storage to allocate to the shared storage pool (as a percentage of total main storage). Contains the null value for private pools defined in subsystem descriptions.
TUNING_MAXIMUM_SIZE	DECIMAL(10,2)	The maximum amount of storage to allocate to the shared storage pool (as a percentage of total main storage). Contains the null value for private pools defined in subsystem descriptions.
TUNING_MINIMUM_FAULTS	DECIMAL(10,2)	The maximum page faults per second to use as a guideline for the shared storage pool. Contains the null value for private pools defined in subsystem descriptions.
TUNING_MAXIMUM_FAULTS	DECIMAL(10,2)	The minimum page faults per second to use as a guideline for the shared storage pool. Contains the null value for private pools defined in subsystem descriptions.
TUNING_THREAD_FAULTS	DECIMAL(10,2)	The page faults per second for each active thread to use as a guideline for the shared storage pool. Contains the null value for private pools defined in subsystem descriptions.

Table 122. MEMORY_POOL table function (continued)

Column Name	Data Type	Description
TUNING_MINIMUM_ACTIVITY	DECIMAL(10,2)	The minimum value that the shared pool's activity level can be set to by the performance adjuster when the QPFRADJ system value is set to 2 or 3. Contains the null value for private pools defined in subsystem descriptions.
TUNING_MAXIMUM_ACTIVITY	DECIMAL(10,2)	The maximum value that the shared pool's activity level can be set to by the performance adjuster when the QPFRADJ system value is set to 2 or 3. Contains the null value for private pools defined in subsystem descriptions.

Example

Return all available pool information, both private and shared, active and inactive. Specify to reset all the elapsed values to 0.

```
SELECT * FROM TABLE(QSYS2.MEMORY_POOL(RESET_STATISTICS=>'YES')) X;
```

MEMORY_POOL_INFO view

The MEMORY_POOL_INFO view returns one row for every active pool.

The information returned is similar to the detail seen from the Work System Status (WRKSYSSTS) command. It does not reset the statistical columns; to do this, use the associated table function, MEMORY_POOL.

The following table describes the columns in the view. The schema is QSYS2.

Table 123. MEMORY_POOL_INFO view

Column Name	System Column Name	Data Type	Description
SYSTEM_POOL_ID	POOL_ID	INTEGER	The system-related pool identifier for each of the system storage pools that currently has main storage allocated to it.
POOL_NAME	POOL_NAME	VARCHAR(10)	The name of this storage pool. The name may be a number, in which case it is a private pool associated with a subsystem, or one of the following special values. ***MACHINE** The specified pool definition is defined to be the machine pool. ***BASE** The specified pool definition is defined to be the base system pool, which can be shared with other subsystems. ***INTERACT** The specified pool definition is defined to be the shared pool used for the QINTER subsystem. ***SPOOL** The specified pool definition is defined to be the shared pool used for spooled writers. ***SHRPOOL*x*** The specified pool definition is defined to be a shared pool.
CURRENT_SIZE	CURR_SIZE	DECIMAL(20,2)	The amount of main storage, in megabytes, in the pool.
RESERVED_SIZE	RSVD_SIZE	DECIMAL(10,2)	The amount of storage, in megabytes, in the pool reserved for system use (for example, for save/restore operations).
DEFINED_SIZE	DFND_SIZE	DECIMAL(20,2)	The size of the pool, in megabytes, as defined in the shared pool, subsystem description, or system value QMCHPOOL. Contains the null value for a pool without a defined size.
MAXIMUM_ACTIVE_THREADS	MAX_THREAD	INTEGER	The maximum number of threads that can be active in the pool at any one time.
CURRENT_THREADS	CURR_THRD	INTEGER	The number of threads currently using the pool.
CURRENT_INELIGIBLE_THREADS	INEL_THRD	INTEGER	The number of ineligible threads in the pool.
SUBSYSTEM_LIBRARY_NAME	SUBLIB_NAM	VARCHAR(10) Nullable	The library containing the subsystem name. Contains the null value for shared pools.

Table 123. MEMORY_POOL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
SUBSYSTEM_NAME	SUB_NAME	VARCHAR(10) Nullable	The subsystem with which this storage pool is associated. Contains the null value for shared pools.
DESCRIPTION	DESC	VARCHAR(50) Nullable	The description of the shared pool. Contains the null value for private pools or if a description does not exist for a shared pool.
PAGING_OPTION	PAGE_OPT	VARCHAR(10)	Whether the system will dynamically adjust the paging characteristics of the storage pool for optimum performance. ***FIXED** The system does not dynamically adjust the paging characteristics. ***CALC** The system dynamically adjusts the paging characteristics. **USRDFN** The system does not dynamically adjust the paging characteristics for the storage pool but uses values that have been defined through the QWCCHGTN API.
ELAPSED_TIME	ELAP_TIME	INTEGER	The time, in seconds, since the measurement start time.
ELAPSED_DATABASE_FAULTS	ELAP_DBF	DECIMAL(10,1)	The rate, in page faults per second, of database page faults against pages containing either database access paths or data.
ELAPSED_NON_DATABASE_FAULTS	ELAP_NDBF	DECIMAL(10,1)	The rate, in page faults per second, of nondatabase page faults against pages other than those designated as database pages.
ELAPSED_TOTAL_FAULTS	ELAP_TOTF	DECIMAL(10,1)	The rate, in page faults per second, of database faults and non-database faults.
ELAPSED_DATABASE_PAGES	ELAP_DBP	DECIMAL(10,1)	The rate, in pages per second, at which database pages are brought into the storage pool.
ELAPSED_NON_DATABASE_PAGES	ELAP_NDBP	DECIMAL(10,1)	The rate in pages per second at which nondatabase pages are brought into the storage pool.
ELAPSED_ACTIVE_TO_WAIT	ELAP_ATW	DECIMAL(10,1)	The rate, in transitions per minute, of transitions of threads from an active condition to a waiting condition.
ELAPSED_WAIT_TO_INELIGIBLE	ELAP_WTI	DECIMAL(10,1)	The rate, in transitions per minute, of transitions of threads from a waiting condition to an ineligible condition.
ELAPSED_ACTIVE_TO_INELIGIBLE	ELAP_ATI	DECIMAL(10,1)	The rate, in transitions per minute, of transitions of threads from an active condition to an ineligible condition.
TUNING_PRIORITY	TUN_PRIOR	INTEGER Nullable	The priority of the shared storage pool used by the system when making automatic performance adjustments. Contains the null value for private pools defined in subsystem descriptions.
TUNING_MINIMUM_SIZE	TUN_MIN_SZ	DECIMAL(10,2) Nullable	The minimum amount of storage to allocate to the shared storage pool (as a percentage of total main storage). Contains the null value for private pools defined in subsystem descriptions.
TUNING_MAXIMUM_SIZE	TUN_MAX_SZ	DECIMAL(10,2) Nullable	The maximum amount of storage to allocate to the shared storage pool (as a percentage of total main storage). Contains the null value for private pools defined in subsystem descriptions.
TUNING_MINIMUM_FAULTS	TUN_MIN_FT	DECIMAL(10,2) Nullable	The maximum page faults per second to use as a guideline for the shared storage pool. Contains the null value for private pools defined in subsystem descriptions.
TUNING_MAXIMUM_FAULTS	TUN_MAX_FT	DECIMAL(10,2) Nullable	The minimum page faults per second to use as a guideline for the shared storage pool. Contains the null value for private pools defined in subsystem descriptions.
TUNING_THREAD_FAULTS	TUN_THR_FT	DECIMAL(10,2) Nullable	The page faults per second for each active thread to use as a guideline for the shared storage pool. Contains the null value for private pools defined in subsystem descriptions.
TUNING_MINIMUM_ACTIVITY	TUN_MIN_AC	DECIMAL(10,2) Nullable	The minimum value that the shared pool's activity level can be set to by the performance adjuster when the QPFRADJ system value is set to 2 or 3. Contains the null value for private pools defined in subsystem descriptions.

Table 123. MEMORY_POOL_INFO view (continued)

Column Name	System Column Name	Data Type	Description
TUNING_MAXIMUM_ACTIVITY	TUN_MAX_AC	DECIMAL(10,2) Nullable	The maximum value that the shared pool's activity level can be set to by the performance adjuster when the QPFRADJ system value is set to 2 or 3. Contains the null value for private pools defined in subsystem descriptions.

Example

Return all active pool information.

```
SELECT * FROM QSYS2.MEMORY_POOL_INFO;
```

OBJECT_LOCK_INFO view

The OBJECT_LOCK_INFO view returns one row for every lock held for every object on the partition in *SYSBAS and in the current thread's ASP group.

The values returned for the columns in the view are closely related to the values returned by Retrieve Lock Information API and Retrieve Lock Request Information API. Refer to the APIs for more detailed information.

When querying this view, you should use a WHERE clause to restrict the result set to avoid excessive use of system resources.

The following table describes the columns in the view. The schema is QSYS2.

Table 124. OBJECT_LOCK_INFO view

Column Name	System Column Name	Data Type	Description
OBJECT_SCHEMA	OSCHEMA	VARCHAR(128)	The name of the schema containing the object.
OBJECT_NAME	NAME	VARCHAR(128)	The name of the object.
SYSTEM_OBJECT_SCHEMA	SYS_DNAME	VARCHAR(10)	The system library name of the object.
SYSTEM_OBJECT_NAME	SYS_ONAME	VARCHAR(10)	The system name of the object
SYSTEM_TABLE_MEMBER	SYS_MNAME	VARCHAR(10) Nullable	The name of the member that is locked in the file. Contains the null value if the lock information is not for a member.
OBJECT_TYPE	OBJTYPE	VARCHAR(8)	The system object type of the locked object.
SQL_OBJECT_TYPE	SQLTYPE	VARCHAR(9) Nullable	The SQL type of the object. Values are: • ALIAS • FUNCTION • INDEX • PACKAGE • PROCEDURE • ROUTINE • SEQUENCE • TABLE • TRIGGER • TYPE • VARIABLE • VIEW • XSR Contains the null value if the object is not an SQL object.
ASP_NUMBER	ASPNUM	INTEGER	The numeric identifier of the ASP containing the object that is locked.
ASPGRP	ASPGRP	VARCHAR(10)	The name of the ASP device containing the object that is locked. Can contain the special value of *SYSBAS.

Table 124. OBJECT_LOCK_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MEMBER_LOCK_TYPE	LOCK_TYPE	VARCHAR(10) Nullable	The type of lock that is held. **MEMBER** Lock on the member control block. **DATA** Lock on the actual data within the member. **ACCESSPATH** Lock on the access path used to access the member's data. Contains the null value if the lock information is not for a member.
LOCK_STATE	LOCK_STATE	VARCHAR(7)	The lock condition for the object or member. ***SHRRD** Lock shared for read. ***SHRUPD** Lock shared for update. ***SHRNUP** Lock shared no update. ***EXCLRD** Lock exclusive allow read. ***EXCL** Lock exclusive no read.
LOCK_STATUS	STATUS	VARCHAR(9)	The status of the lock. **HELD** The lock is currently held by the job. **WAITING** The job is waiting for the lock. **REQUESTED** The job has a lock request outstanding for the object.
LOCK_SCOPE	LOCK_SCOPE	VARCHAR(10)	The scope of the lock. Values are: • JOB • THREAD • LOCK SPACE
JOB_NAME	JOB_NAME	VARCHAR(28)	The qualified job name.
THREAD_ID	THREAD_ID	BIGINT Nullable	The thread that is associated with the lock. • If a held lock is job scoped, returns the null value. If a held lock is thread scoped, contains the identifier for the thread holding the lock. • If the scope of the lock is to the lock space and the lock is not held, contains the identifier of the thread requesting the lock. • If the lock is requested but not yet available, contains the identifier of the thread requesting the lock.
LOCK_SPACE_ID	LOCKID	BINARY(20) Nullable	When the LOCK_SCOPE column value is LOCK SPACE and the lock is being waited on by a thread, contains the lock space ID value for which the lock is being waited on. Otherwise, contains the null value.
LOCK_COUNT	LOCK_COUNT	INTEGER	The number of identical locks held.
PROGRAM_LIBRARY_NAME	PROGLIB	VARCHAR(10) Nullable	The name of the library containing the program or service program. Contains the null value if the lock holder information is not available.
PROGRAM_NAME	PROGNAME	VARCHAR(10) Nullable	The name of the program holding the lock. This can be any type of program object, including objects of type *PGM and *SRVPGM. Contains the null value if the lock holder information is not available.
MODULE_LIBRARY	MODLIB	VARCHAR(10) Nullable	The library containing the module. Contains the null value if the lock holder information is not available or if the program is not an ILE program.

Table 124. OBJECT_LOCK_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MODULE_NAME	MODNAME	VARCHAR(10) Nullable	The module containing the ILE procedure. Contains the null value if the lock holder information is not available or if the program is not an ILE program.
PROCEDURE_NAME	PROCNAME	VARCHAR(10) Nullable	The name of the procedure. Contains the null value if the lock holder information is not available.
STATEMENT_ID	STMTID	CHAR(10) Nullable	The high-level language statement identifier. For a character representation of a number, the number is right-adjusted and padded on the left with zeros (for example, '0000000246'). Contains the null value if the lock holder information is not available.
MACHINE_INSTRUCTION	INSTRUCT	INTEGER Nullable	The current machine instruction number in the program. Contains the null value if the lock holder information is not available or if it is an ILE procedure.

Example

Find all the jobs holding object locks over the SALES table:

```
SELECT * FROM QSYS2.OBJECT_LOCK_INFO
   WHERE SYSTEM_OBJECT_NAME = 'SALES'
```

RECORD_LOCK_INFO view

The RECORD_LOCK_INFO view returns one row for every record lock for the partition.

The values returned for the columns in the view are closely related to the values returned by Retrieve Record Locks API. Refer to the APIs for more detailed information.

When querying this view, you should use a WHERE clause to restrict the result set to avoid excessive use of system resources.

The following table describes the columns in the view. The schema is QSYS2.

Table 125. RECORD_LOCK_INFO view

Column Name	System Column Name	Data Type	Description
TABLE_SCHEMA	TABSCHEMA	VARCHAR(128)	Name of the schema.
TABLE_NAME	TABNAME	VARCHAR(128)	Name of the table.
TABLE_PARTITION	TABPART	VARCHAR(128)	Name of the table partition or member that contains the locked record.
SYSTEM_TABLE_SCHEMA	SYS_DNAME	VARCHAR(10)	System name of the schema.
SYSTEM_TABLE_NAME	SYS_TNAME	VARCHAR(10)	System name of the table
SYSTEM_TABLE_MEMBER	SYS_MNAME	VARCHAR(10)	The name of the member that contains the locked record.
RELATIVE_RECORD_NUMBER	RRN	BIGINT	The relative record number (RRN) of the record that is locked.

Table 125. RECORD_LOCK_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LOCK_STATE	LOCK_STATE	VARCHAR(8)	The lock condition for the record. **READ** The record is locked for read. Another job may read the same record but cannot lock the record for update intent. The record cannot be changed by another job as long as one job holds a read lock on the record. **UPDATE** The record is locked for update intent. Another job may read the record but may not obtain a read or update lock on it until the lock is released. **INTERNAL** The row is locked internally for read. For a short time the operating system holds an internal lock to access the row. Another job may read the same row and may even have the row locked for update intent. However, if another job does have the row locked for update intent, the actual change of the row will not proceed until the internal lock is released.
LOCK_STATUS	STATUS	VARCHAR(9)	The status of the lock. **HELD** The lock is currently held by the job. **WAITING** The job is waiting for the lock.
LOCK_SCOPE	LOCK_SCOPE	VARCHAR(10)	The scope of the lock. Values are: • JOB • THREAD • LOCK SPACE
JOB_NAME	JOB_NAME	VARCHAR(28)	The qualified job name.
THREAD_ID	THREAD_ID	BIGINT Nullable	The thread that is associated with the lock. • If a held lock is job scoped, returns the null value. If a held lock is thread scoped, contains the identifier for the thread holding the lock. • If the scope of the lock is to the lock space and the lock is not held, contains the identifier of the thread requesting the lock. • If the lock is requested but not yet available, contains the identifier of the thread requesting the lock.
LOCK_SPACE_ID	LOCKID	BINARY(20) Nullable	When the LOCK_SCOPE column value is LOCK SPACE and the lock is being waited on by a thread, contains the lock space ID value for which the lock is being waited on. Otherwise, contains the null value.

Example

Review the jobs that are updating the SALES table:

```
SELECT JOB_NAME, COUNT(*) AS ROWS_UPDATING
  FROM QSYS2.RECORD_LOCK_INFO
  WHERE SYSTEM_TABLE_NAME = 'SALES' AND
        SYSTEM_TABLE_SCHEMA = 'TOYSTORE' AND
        LOCK_STATE = 'UPDATE'
  GROUP BY JOB_NAME
  ORDER BY ROWS_UPDATING DESC
```

SCHEDULED_JOB_INFO view

The SCHEDULED_JOB_INFO view returns information that can also be seen through the Work with Job Schedule Entries (WRKJOBSCDE) command interface. Each job schedule entry contains the information to automatically submit a batch job once or at regularly scheduled intervals.

Authorization: No authority is required to access scheduled job rows, but some columns return NULL if you don't have the required authority. You must have *JOBCTL special authority or be the user profile listed in the SCHEDULED_BY column to see the data in all columns.

The following table describes the columns in the view. The schema is QSYS2.

Table 126. SCHEDULED_JOB_INFO view

Column Name	System Column Name	Data Type	Description
SCHEDULED_JOB_ENTRY_NUMBER	ENTRYNO	INTEGER	The number assigned to the job schedule entry when the entry is added to the job schedule.
SCHEDULED_JOB_NAME	SCDJOBNAME	VARCHAR(10)	The name of the job schedule entry. This is the simple job name portion of the fully qualified job name used when the job is submitted. It is also used to identify the job schedule entry through change, hold, release and remove functions.
SCHEDULED_DATE_VALUE	SCDDATEV	VARCHAR(14)	Indicates the date on which the job is scheduled to be submitted. **SCHEDULED_DATE** The date in the SCHEDULED_DATE column is used **SCHEDULED_DAYS** The days in the SCHEDULED_DAYS column are used ***MONTHSTR** The first day of the month is used. ***MONTHEND** The last day of the month is used.
SCHEDULED_DATE	SCDDATE	DATE Nullable	The date on which the job is scheduled to be submitted. Contains the null value if the SCHEDULED_DATE_VALUE column is not SCHEDULED_DATE.
SCHEDULED_TIME	SCDTIME	TIME	The time when the job is scheduled to be submitted on the scheduled date.
SCHEDULED_DAYS	SCDDAYS	VARCHAR(34) Nullable	The days on which the job is submitted if a specific date is not specified. The value is a comma separated string with any or all of the values: *MON *TUE *WED *THU *FRI *SAT *SUN. The single value of *ALL can be returned to represent all seven values. Contains the null value if SCHEDULED_DATE_VALUE is not SCHEDULED_DAYS.
FREQUENCY	FREQUENCY	VARCHAR(8)	How often the job is to be submitted. ***ONCE** The job is scheduled to be submitted a single time. ***WEEKLY** The job is scheduled to be submitted on the same day or days of each week at the scheduled time. ***MONTHLY** The job is scheduled to be submitted on the same day or days of each month at the scheduled time.

Table 126. SCHEDULED_JOB_INFO view (continued)

Column Name	System Column Name	Data Type	Description
RELATIVE_DAYS_OF_MONTH	RELDAYSMON	VARCHAR(13) Nullable	Specifies which occurrence during the month (for the days listed in the SCHEDULED_DAYS column) the job is scheduled to be run. The value is a comma separated string with up to five of the following values: **1** The job is scheduled for the first occurrence of the day or days (SCHEDULED_DAYS: *MON and *WED for example) of the month. **2** The job is scheduled for the second occurrence of the day or days of the month. **3** The job is scheduled for the third occurrence of the day or days of the month. **4** The job is scheduled for the fourth occurrence of the day or days of the month. **5** The job is scheduled for the fifth occurrence of the day or days of the month. ***LAST** The job is scheduled for the last occurrence of the day or days of the month. Contains the null value if the FREQUENCY column does not have a value of MONTHLY or SCHEDULED_DAYS is null.
RECOVERY_ACTION	RECOVERY	VARCHAR(7)	The recovery action taken when the system is powered down or in the restricted state at the time a job is scheduled to be submitted. ***SBMRLS** Submit a job to the job queue as a released job. ***SBMHLD** Submit a job to the job queue as a held job. ***NOSBM** Do not submit a job to the job queue.
NEXT_SUBMISSION_DATE	NXTSUBDATE	DATE Nullable	The next date that the job scheduling process is scheduled to submit this job. Contains the null value if the job is not scheduled to be submitted again.
STATUS	STATUS	VARCHAR(9)	The status of the job schedule entry. **HELD** The entry is held. If an entry has a status of HELD at the scheduled date and time, a job is not submitted. **SAVED** The entry is defined with a frequency of ONCE and a save value of *YES at a time later than the scheduled date and time. **SCHEDULED** The entry is waiting until the scheduled date and time for a job to be submitted.
JOB_QUEUE_NAME	JOBQ	VARCHAR(10)	The job queue to which the job is scheduled to be submitted. Can contain the special value of *JOBD, meaning that the job is submitted to the job queue specified in the job description listed in the JOB_DESCRIPTION_NAME and JOB_DESCRIPTION_LIBRARY_NAME columns.
JOB_QUEUE_LIBRARY_NAME	JOBQLIB	VARCHAR(10) Nullable	The library containing the job queue. Contains the null value if JOB_QUEUE_NAME is *JOBD .

Table 126. SCHEDULED_JOB_INFO view (continued)

Column Name	System Column Name	Data Type	Description
JOB_QUEUE_STATUS	JOBQSTATUS	VARCHAR(10) Nullable	The status of the job queue. **HLD** The job queue is held, but not attached to an active subsystem. **HLD/SBS** The job queue is held and attached to an active subsystem. **LOCKED** The status of the job queue could not be determined because a lock could not be obtained on the job queue. **RLS** The job queue is released, but not attached to an active subsystem. **RLS/SBS** The job queue is released and attached to an active subsystem. Contains the null value if JOB_QUEUE_NAME is *JOBD, if the job queue is not found or is damaged, or if the information is not available.
DATES_OMITTED	OMITDATES	VARCHAR(219) Nullable	A comma separated string with up to 20 dates in *ISO format indicating dates when the job will not be scheduled to run. Contains the null value if no dates were specified to omit or if the information is not available.
SCHEDULED_BY	CREATEDBY	VARCHAR(10)	The user profile of the job which added the entry to the job schedule.
DESCRIPTION	TEXT	VARCHAR(50) Nullable	The descriptive text for the job schedule entry. Contains the null value if the job schedule entry has no description.
COMMAND_STRING	COMMAND	VARCHAR(512) Nullable	The command that is run in the submitted job. Contains the null value if the information is not available.
USER_PROFILE_FOR_SUBMITTED_JOB	SBMJOBUSR	VARCHAR(10) Nullable	The user profile to be used when the job is submitted. Can contain the special value *JOBD to indicate that the user profile from the job description is used. Contains the null value if the information is not available.
JOB_DESCRIPTION_NAME	JOBD	VARCHAR(10) Nullable	The job description used when the job is submitted. Can contain the special value of *USRPRF to indicate that the job description specified in the user profile under which the submitted job runs is used. Contains the null value if the information is not available.
JOB_DESCRIPTION_LIBRARY_NAME	JOBDLIB	VARCHAR(10) Nullable	The library containing the job description. Contains the null value if JOB_DESCRIPTION_NAME has a value of *USRPRF or if the information is not available.
MESSAGE_QUEUE_NAME	MSGQ	VARCHAR(10) Nullable	The name of the message queue where the messages for this job schedule entry are sent. Can contain the special value *USRPRF to indicate that the message queue specified in the user profile under which the submitted job runs is used. Contains the null value is no specific message queue is associated with this job schedule entry or if the information is not available.
MESSAGE_QUEUE_LIBRARY_NAME	MSGQLIB	VARCHAR(10) Nullable	The library containing the message queue. Contains the null value if MESSAGE_QUEUE_NAME is null, contains the special value of *USRPRF, or if the information is not available.
LAST_SUCCESSFUL_SUBMISSION_ TIMESTAMP	SBMTIMSTMP	TIMESTAMP(0) Nullable	The timestamp when a batch job was last successfully submitted for the job schedule entry. Contains the null value if the job schedule entry has not been used to submit a job.
LAST_SUCCESSFUL_SUBMISSION_JOB	LASTSBMJOB	VARCHAR(28) Nullable	The qualified job name used when this scheduled job was last submitted. Contains the null value if the scheduled job has never been submitted or if the information is not available.

Table 126. SCHEDULED_JOB_INFO view (continued)

Column Name	System Column Name	Data Type	Description
LAST_ATTEMPTED_SUBMISSION_ TIMESTAMP	ATTSBMTIM	TIMESTAMP(0) Nullable	The timestamp when this scheduled job was last submitted. Contains the null value if the scheduled job has never been submitted or if the information is not available.
LAST_ATTEMPTED_SUBMISSION_STATUS	SBMJOBSTS	VARCHAR(68) Nullable	The status from when this scheduled job was last submitted. Values are: • JOB SUCCESSFULLY SUBMITTED • LAST JOB SUBMISSION FAILED, CHECK THE JOB MESSAGE QUEUE FOR DETAILS • JOB NOT SUBMITTED DUE TO HELD STATUS • JOB SUBMITTED AFTER SCHEDULED TIME AS SPECIFIED BY RECOVERY ACTION • JOB NOT SUBMITTED AS SPECIFIED BY RECOVERY ACTION Contains the null value if the scheduled job has never been submitted or if the information is not available.
KEEP_ENTRY	KEEP	VARCHAR(3) Nullable	Whether the job schedule entry is kept or removed after the job has been submitted. **YES** The job schedule entry is kept. **NO** The job schedule entry is removed. Contains the null value when the FREQUENCY column does not contain *ONCE or if the information is not available.

Example

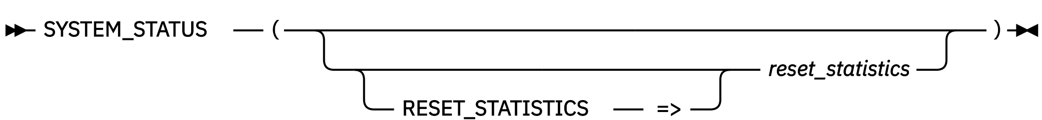
Review the job scheduled entries which are no longer in effect, either because they were explicitly held or because they were scheduled to run a single time and the scheduled date and time has passed.

```
SELECT * FROM QSYS2.SCHEDULED_JOB_INFO WHERE STATUS IN ('HELD', 'SAVED')
  ORDER BY SCHEDULED_BY;
```

SYSTEM_STATUS table function

The SYSTEM_STATUS table function returns a single row containing details about the current partition.

The information returned is similar to the detail seen from the Work with System Status (WRKSYSSTS) and the Work with System Activity (WRKSYSACT) commands.



The schema is QSYS2.

reset_statistics

A character or graphic string expression that contains a value of YES or NO.

If this parameter has a value of YES, statistics are reset such that the time of this query execution is used as the new baseline. The columns that contain this statistical data have names that are prefixed with ELAPSED_. Future invocations of SYSTEM_STATUS within this connection will return statistical detail relative to the new baseline. If this parameter has a value of NO, statistics are not reset for the invocation. If this parameter is not specified, the default is NO.

The result of the function is a table containing multiple rows with the format shown in the following table. All the columns are nullable.

Table 127. SYSTEM_STATUS table function

Column Name	Data Type	Description
TOTAL_JOBS_IN_SYSTEM	INTEGER	The total number of user and system jobs that are currently in the system. The total includes: • All jobs on job queues waiting to be processed. • All jobs currently active (being processed). • All jobs that have completed running but still have output on output queues to be produced.
MAXIMUM_JOBS_IN_SYSTEM	INTEGER	The maximum number of jobs that are allowed on the system. When the number of jobs reaches this maximum, you can no longer submit or start more jobs on the system. The total includes: • All jobs on job queues waiting to be processed. • All jobs currently active (being processed). • All jobs that have completed running but still have output on output queues to be produced.
ACTIVE_JOBS_IN_SYSTEM	INTEGER	The number of jobs active in the system (jobs that have been started, but have not yet ended), including both user and system jobs.
INTERACTIVE_JOBS_IN_SYSTEM	DECIMAL(10,2)	The percentage of interactive performance assigned to this logical partition. This value is a percentage of the total interactive performance available to the entire physical system.
ELAPSED_TIME	INTEGER	The time that has elapsed, in seconds, between the measurement start time and the current system time.
ELAPSED_CPU_USED	DECIMAL(10,2)	The average of the elapsed time during which the processing units were in use.
ELAPSED_CPU_SHARED	DECIMAL(10,2)	The percentage of the total shared processor pool capacity used by all partitions using the pool during the elapsed time. Returns null if this partition does not share processors.
ELAPSED_CPU_UNCAPPED_CAPACITY	DECIMAL(10,2)	The percentage of the uncapped shared processing capacity for the partition used since the last time statistics were reset. Returns null if this partition cannot use more that its configured processing capacity.
CONFIGURED_CPUS	INTEGER	Total number of configured CPUs for the partition.
CPU_SHARING_ATTRIBUTE	VARCHAR(8)	This attribute indicates whether this partition is sharing processors. If the value indicates the partition does not share physical processors, then this partition uses only dedicated processors. If the value indicates the partition shares physical processors, then this partition uses physical processors from a shared pool of physical processors. **CAPPED** Partition shares processors. The partition is limited to using its configured capacity. **UNCAPPED** Partition shares processors. The partition can use more than its configured capacity. Contains the null value if the partition does not share processors.
CURRENT_CPU_CAPACITY	DECIMAL(10,2)	The current processing capacity specifies the processor units that are being used in the partition. For a partition sharing physical processors, the current processing capacity represents the share of the physical processors in the pool it is running. For a partition using dedicated processors, the current processing capacity represents the number of virtual processors that are currently active in the partition.
AVERAGE_CPU_RATE	DECIMAL(20,2)	The average CPU rate expressed as a percentage where 100% indicates the processor is running at its nominal frequency. A value above or below 100% indicates how much the processor has been slowed down (throttled) or speeded up (turbo) relative to the nominal frequency for the processor model. For instance, a value of 120% indicates the processor is running 20% faster against its nominal speed.
AVERAGE_CPU_UTILIZATION	DECIMAL(20,2)	The average CPU utilization for all the active processors.
MINIMUM_CPU_UTILIZATION	DECIMAL(20,2)	The CPU utilization of the processor that reported the minimum amount of CPU utilization.
MAXIMUM_CPU_UTILIZATION	DECIMAL(20,2)	The CPU utilization of the processor that reported the maximum amount of CPU utilization.
SQL_CPU_UTILIZATION	DECIMAL(20,2)	Always contains the null value.

Table 127. SYSTEM_STATUS table function (continued)

Column Name	Data Type	Description
MAIN_STORAGE_SIZE	BIGINT	The amount of main storage, in kilobytes, in the system.
SYSTEM_ASP_STORAGE	BIGINT	The storage capacity of the system auxiliary storage pool (ASP number 1) in millions of bytes. This value represents the amount of space available for storage of both permanent and temporary objects.
TOTAL_AUXILIARY_STORAGE	BIGINT	The total auxiliary storage, in millions of bytes, on the system.
SYSTEM_ASP_USED	DECIMAL(10,2)	The percentage of the system storage pool (ASP number 1) currently in use.
CURRENT_TEMPORARY_STORAGE	INTEGER	The current amount of storage, in millions of bytes, in use for temporary objects.
MAXIMUM_TEMPORARY_STORAGE_USED	INTEGER	The largest amount of storage, in millions of bytes, used for temporary objects at any one time since the last IPL.
PERMANENT_ADDRESS_RATE	DECIMAL(6,3)	The percentage of the maximum possible addresses for permanent objects that have been used.
TEMPORARY_ADDRESS_RATE	DECIMAL(6,3)	The percentage of the maximum possible addresses for temporary objects that have been used.
TEMPORARY_256MB_SEGMENTS	DECIMAL(10,2)	The percentage of the maximum possible temporary 256MB segments that have been used.
TEMPORARY_4GB_SEGMENTS	DECIMAL(10,2)	The percentage of the maximum possible temporary 4GB segments that have been used.
PERMANENT_256MB_SEGMENTS	DECIMAL(10,2)	The percentage of the maximum possible permanent 256MB segments that have been used.
PERMANENT_4GB_SEGMENTS	DECIMAL(10,2)	The percentage of the maximum possible permanent 4GB segments that have been used.
HOST_NAME	VARCHAR(255)	Name of the system where this information was generated.
PARTITION_ID	INTEGER	The identifier for the partition in which this view is being run.
NUMBER_OF_PARTITIONS	INTEGER	The number of partitions on the system. This includes partitions that are currently powered on (running) and partitions that are powered off.
ACTIVE_THREADS_IN_SYSTEM	INTEGER	The number of initial and secondary threads in the system (threads that have been started, but have not yet ended), including both user and system threads.
RESTRICTED_STATE	VARCHAR(3)	Whether the system is in restricted state. **NO** System is not in restricted state. **YES** System is in restricted state.

Example

Return storage and CPU status for the partition. Specify to reset all the elapsed values to 0.

```
SELECT * FROM TABLE(QSYS2.SYSTEM_STATUS(RESET_STATISTICS=>'YES')) X;
```

SYSTEM_STATUS_INFO view

The SYSTEM_STATUS_INFO view returns a single row containing details about the current partition.

The information returned is similar to the detail seen from the Work with System Status (WRKSYSSTS) and the Work with System Activity (WRKSYSACT) commands. It does not reset the statistical columns; to do this, use the associated table function, “SYSTEM_STATUS table function” on page 485.

The following table describes the columns in the view. The schema is QSYS2.

Table 128. SYSTEM_STATUS_INFO view

Column Name	System Column Name	Data Type	Description
TOTAL_JOBS_IN_SYSTEM	TOTAL_JOBS	INTEGER	The total number of user and system jobs that are currently in the system. The total includes: • All jobs on job queues waiting to be processed. • All jobs currently active (being processed). • All jobs that have completed running but still have output on output queues to be produced.
MAXIMUM_JOBS_IN_SYSTEM	MAX_JOBS	INTEGER	The maximum number of jobs that are allowed on the system. When the number of jobs reaches this maximum, you can no longer submit or start more jobs on the system. The total includes: • All jobs on job queues waiting to be processed. • All jobs currently active (being processed). • All jobs that have completed running but still have output on output queues to be produced.
ACTIVE_JOBS_IN_SYSTEM	ACT_JOBS	INTEGER	The number of jobs active in the system (jobs that have been started, but have not yet ended), including both user and system jobs.
INTERACTIVE_JOBS_IN_SYSTEM	INTER_JOBS	DECIMAL(5,2)	The percentage of interactive performance assigned to this logical partition. This value is a percentage of the total interactive performance available to the entire physical system.
ELAPSED_TIME	ELAP_TIME	INTEGER	The time that has elapsed, in seconds, between the measurement start time and the current system time.
ELAPSED_CPU_USED	ELAP_USED	DECIMAL(5,2)	The average of the elapsed time during which the processing units were in use.
ELAPSED_CPU_SHARED	ELAP_SHARE	DECIMAL(5,2) Nullable	The percentage of the total shared processor pool capacity used by all partitions using the pool during the elapsed time. Returns null if this partition does not share processors.
ELAPSED_CPU_UNCAPPED_ CAPACITY	ELAP_UNCAP	DECIMAL(5,2) Nullable	The percentage of the uncapped shared processing capacity for the partition used since the last time statistics were reset. Returns null if this partition cannot use more that its configured processing capacity.
CONFIGURED_CPUS	CONFIGCPUS	INTEGER	Total number of configured CPUs for the partition.
CPU_SHARING_ATTRIBUTE	CPU_SHARE	VARCHAR(8) Nullable	This attribute indicates whether this partition is sharing processors. If the value indicates the partition does not share physical processors, then this partition uses only dedicated processors. If the value indicates the partition shares physical processors, then this partition uses physical processors from a shared pool of physical processors. **CAPPED** Partition shares processors. The partition is limited to using its configured capacity. **UNCAPPED** Partition shares processors. The partition can use more than its configured capacity. Contains the null value if the partition does not share processors.
CURRENT_CPU_CAPACITY	CPU_CAP	DECIMAL(5,2)	The current processing capacity specifies the processor units that are being used in the partition. For a partition sharing physical processors, the current processing capacity represents the share of the physical processors in the pool it is running. For a partition using dedicated processors, the current processing capacity represents the number of virtual processors that are currently active in the partition.
AVERAGE_CPU_RATE	CPU_RATE	DECIMAL(5,2)	The average CPU rate expressed as a percentage where 100% indicates the processor is running at its nominal frequency. A value above or below 100% indicates how much the processor has been slowed down (throttled) or speeded up (turbo) relative to the nominal frequency for the processor model. For instance, a value of 120% indicates the processor is running 20% faster against its nominal speed.
AVERAGE_CPU_UTILIZATION	CPU_AVG	DECIMAL(5,2)	The average CPU utilization for all the active processors.
MINIMUM_CPU_UTILIZATION	CPU_MIN	DECIMAL(5,2)	The CPU utilization of the processor that reported the minimum amount of CPU utilization.

Table 128. SYSTEM_STATUS_INFO view (continued)

Column Name	System Column Name	Data Type	Description
MAXIMUM_CPU_UTILIZATION	CPU_MAX	DECIMAL(5,2)	The CPU utilization of the processor that reported the maximum amount of CPU utilization.
SQL_CPU_UTILIZATION	CPU_SQL	DECIMAL(5,2) Nullable	Always contains the null value.
MAIN_STORAGE_SIZE	MAIN_STG	BIGINT	The amount of main storage, in kilobytes, in the system.
SYSTEM_ASP_STORAGE	SYS_STG	BIGINT	The storage capacity of the system auxiliary storage pool (ASP number 1) in millions of bytes. This value represents the amount of space available for storage of both permanent and temporary objects.
TOTAL_AUXILIARY_STORAGE	AUX_STG	BIGINT	The total auxiliary storage, in millions of bytes, on the system.
SYSTEM_ASP_USED	SYS_RATE	DECIMAL(5,2)	The percentage of the system storage pool (ASP number 1) currently in use.
CURRENT_TEMPORARY_STORAGE	TEMP_CUR	INTEGER	The current amount of storage, in millions of bytes, in use for temporary objects.
MAXIMUM_TEMPORARY_STORAGE_ USED	TEMP_MAX	INTEGER	The largest amount of storage, in millions of bytes, used for temporary objects at any one time since the last IPL.
PERMANENT_ADDRESS_RATE	PERM_RATE	DECIMAL(6,3)	The percentage of the maximum possible addresses for permanent objects that have been used.
TEMPORARY_ADDRESS_RATE	TEMP_RATE	DECIMAL(6,3)	The percentage of the maximum possible addresses for temporary objects that have been used.
TEMPORARY_256MB_SEGMENTS	TEMP_256MB	DECIMAL(5,2)	The percentage of the maximum possible temporary 256MB segments that have been used.
TEMPORARY_4GB_SEGMENTS	TEMP_4GB	DECIMAL(5,2)	The percentage of the maximum possible temporary 4GB segments that have been used.
PERMANENT_256MB_SEGMENTS	PERM_256MB	DECIMAL(5,2)	The percentage of the maximum possible permanent 256MB segments that have been used.
PERMANENT_4GB_SEGMENTS	PERM_4GB	DECIMAL(5,2)	The percentage of the maximum possible permanent 4GB segments that have been used.
HOST_NAME	HOST_NAME	VARCHAR(255)	Name of the system where this information was generated.
PARTITION_ID	PART_ID	INTEGER	The identifier for the partition in which this view is being run.
NUMBER_OF_PARTITIONS	NUM_PART	INTEGER	The number of partitions on the system. This includes partitions that are currently powered on (running) and partitions that are powered off.
ACTIVE_THREADS_IN_SYSTEM	ACT_THREAD	INTEGER	The number of initial and secondary threads in the system (threads that have been started, but have not yet ended), including both user and system threads.
RESTRICTED_STATE	REST_STATE	VARCHAR(3)	Whether the system is in restricted state. **NO** System is not in restricted state. **YES** System is in restricted state.

Example

Review the storage and CPU status for the partition.

```
SELECT * FROM QSYS2.SYSTEM_STATUS_INFO;
```

SYSTEM_VALUE_INFO view

The SYSTEM_VALUE_INFO view contains information about system values.

This view returns the names of system values and their values. The list of system values can be found in Retrieve System Values (QWCRSVAL) API.

*ALLOBJ or *AUDIT special authority is required to retrieve the values for QAUDCTL, QAUDENDACN, QAUDFRCLVL, QAUDLVL, QAUDLVL2, and QCRTOBJAUD. The current value column will contain '*NOTAVL' or -1 when accessed by an unauthorized user.

The following table describes the columns in the view. The schema is QSYS2.

Table 129. SYSTEM_VALUE_INFO view

Column Name	System Column Name	Data Type	Description
SYSTEM_VALUE_NAME	SYSVALNAME	VARCHAR(10)	Name of the system value.
CURRENT_NUMERIC_VALUE	CURNUMVAL	BIGINT	Contains the value if the system value is numeric data. Otherwise, contains the null value.
CURRENT_CHARACTER_VALUE	CURCHARVAL	VARGRAPHIC(1280) CCSID(1200)	Contains the value if the system value is character data. Otherwise, contains the null value.

Example

Look at the system values related to maximums.

```
SELECT * FROM SYSTEM_VALUE_INFO
WHERE SYSTEM_VALUE_NAME LIKE '%MAX%'
```

returns

```
 SYSTEM_VALUE_NAME  CURRENT_NUMERIC_VALUE   CURRENT_CHARACTER_VALUE
   QMAXACTLVL                    32,767      -
   QMAXSIGN                           -      000005
   QPWDMAXLEN                         8      -
   QMAXSGNACN                         -      3
   QMAXJOB                      163,520      -
   QMAXSPLF                       9,999      -
```

SYSTOOLS

SYSTOOLS is a set of DB2 for IBM i supplied examples and tools.

SYSTOOLS is the name of a Database supplied schema (library). SYSTOOLS differs from other DB2 for i supplied schemas (QSYS, QSYS2, SYSIBM, and SYSIBMADM) in that it is not part of the default system path. As general purpose useful tools or examples are built by IBM, they are considered for inclusion within SYSTOOLS. SYSTOOLS provides a wider audience with the opportunity to extract value from the tools.

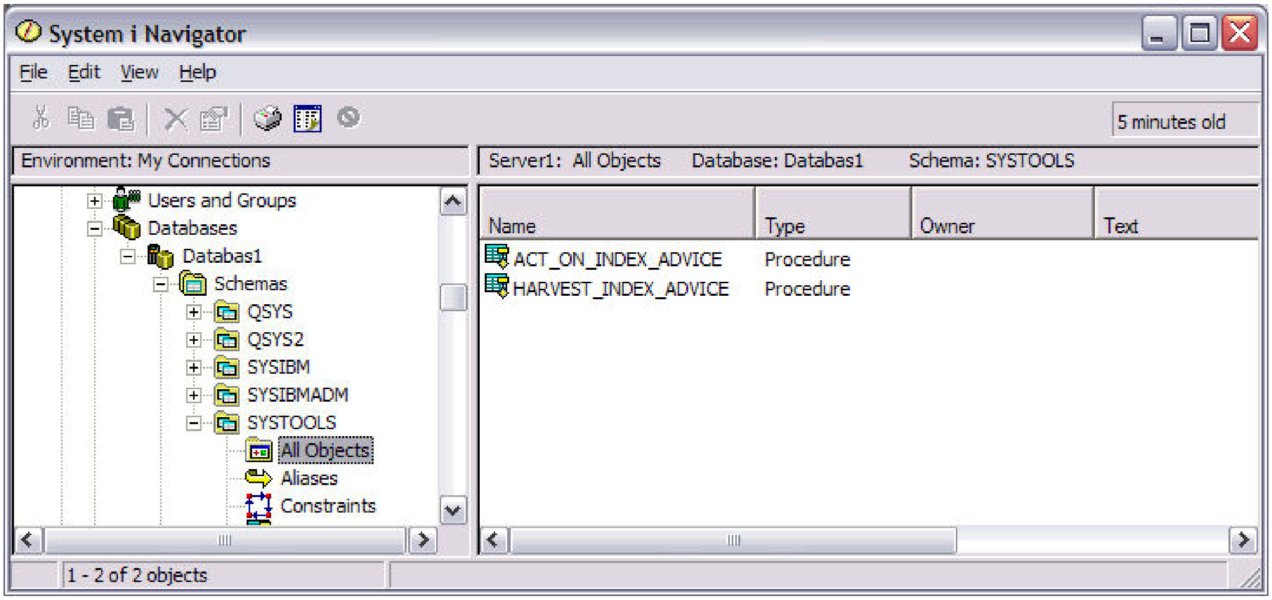
It is the intention of IBM to add content dynamically to SYSTOOLS, either on base releases or through PTFs for field releases. A best practice for customers who are interested in such tools would be to periodically review the contents of SYSTOOLS.

Using SYSTOOLS

You can generate the sample SQL procedures, learn how to call the procedures, and understand the outcome that is expected. You can also modify the procedure source to customize an example into your business operations.

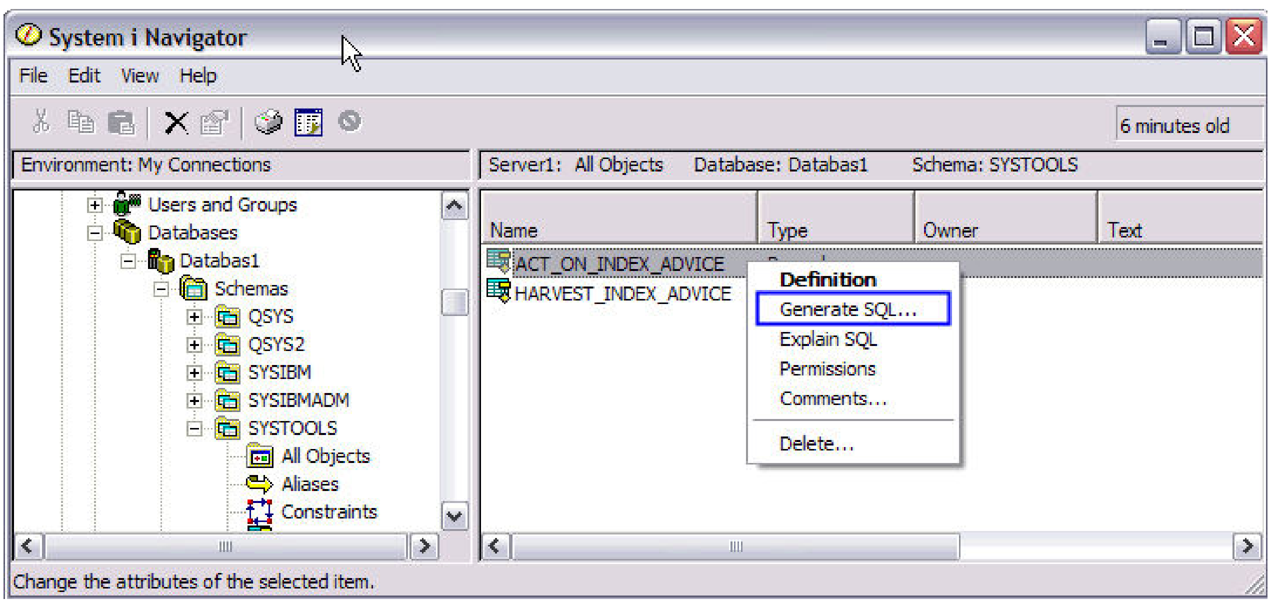
Use System i Navigator, as shown in Figure 1.

Figure 1. System i Navigator schema view of SYSTOOLS:



Start with the Generate SQL action, as shown in Figure 2, to discover and learn within SYSTOOLS. This action utilizes the Generate Data Definition Language (QSQGNDDL) API to produce the CREATE PROCEDURE (SQL) statement. This statement is needed to create a replica of the IBM supplied procedure.

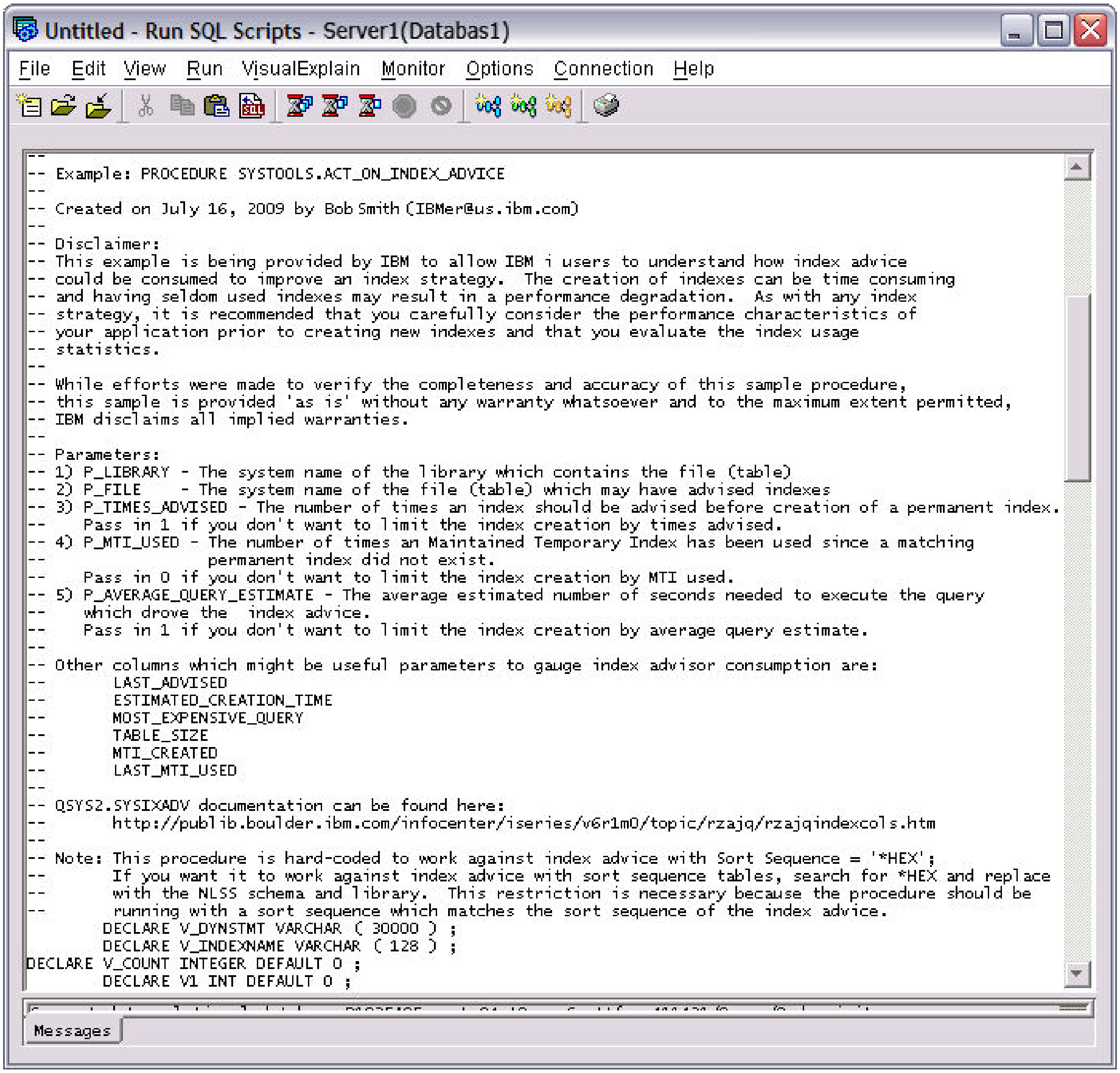
Figure 2. Launching Generate SQL from System i Navigator:



After the Generate SQL action completes, as shown in figure 3, you will have a Run SQL Scripts window active, allowing you to do the following:

1. Scroll down and read the procedure prolog.
2. Understand how to call the procedure and the outcome that is expected.
3. Modify the procedure source, including the procedure name and schema. This capability could be the most useful aspect of SYSTOOLS, allowing you to quickly claim and customize an IBM supplied example into your business operations.

Figure 3. Run SQL Scripts view of the generated SQL:



The IBM maintenance of SYSTOOLS includes periodically dropping and recreating the IBM supplied objects. Customers are allowed to create their own objects within SYSTOOLS. However, if your user created objects conflict with the IBM supplied objects, your objects might be deleted. The tools and examples within SYSTOOLS are considered ready for use. However, they are not subject to IBM Service and Support as they are not considered part of any IBM product.

Database monitor formats

This section contains the formats used to create the database monitor SQL tables and views.

Database monitor SQL table format

Displays the format used to create the QSYS/QAQQDBMN performance statistics table, that is shipped with the system.

```
CREATE TABLE QSYS.QAQQDBMN (
  QQRID DECIMAL(15, 0) NOT NULL DEFAULT 0 ,
  QQTIME TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
  QQJFLD CHAR(46) CCSID 65535 NOT NULL DEFAULT '' ,
  QQRDBN CHAR(18) CCSID 37 NOT NULL DEFAULT '' ,
```

```
     QQSYS CHAR(8) CCSID 37 NOT NULL DEFAULT '' ,
     QQJOB CHAR(10) CCSID 37 NOT NULL DEFAULT '' ,
     QQUSER CHAR(10) CCSID 37 NOT NULL DEFAULT '' ,
     QQJNUM CHAR(6) CCSID 37 NOT NULL DEFAULT '' ,
     QQUCNT DECIMAL(15, 0) DEFAULT NULL ,
     QQUDEF VARCHAR(100) CCSID 37 DEFAULT NULL ,
     QQSTN DECIMAL(15, 0) DEFAULT NULL ,
     QQQDTN DECIMAL(15, 0) DEFAULT NULL ,
     QQQDTL DECIMAL(15, 0) DEFAULT NULL ,
     QQMATN DECIMAL(15, 0) DEFAULT NULL ,
     QQMATL DECIMAL(15, 0) DEFAULT NULL ,
     QQTLN CHAR(10) CCSID 37 DEFAULT NULL ,
     QQTFN CHAR(10) CCSID 37 DEFAULT NULL ,
     QQTMN CHAR(10) CCSID 37 DEFAULT NULL ,
     QQPTLN CHAR(10) CCSID 37 DEFAULT NULL ,
     QQPTFN CHAR(10) CCSID 37 DEFAULT NULL ,
     QQPTMN CHAR(10) CCSID 37 DEFAULT NULL ,
     QQILNM CHAR(10) CCSID 37 DEFAULT NULL ,
     QQIFNM CHAR(10) CCSID 37 DEFAULT NULL ,
     QQIMNM CHAR(10) CCSID 37 DEFAULT NULL ,
     QQNTNM CHAR(10) CCSID 37 DEFAULT NULL ,
     QQNLNM CHAR(10) CCSID 37 DEFAULT NULL ,
     QQSTIM TIMESTAMP DEFAULT NULL ,
     QQETIM TIMESTAMP DEFAULT NULL ,
     QQKP CHAR(1) CCSID 37 DEFAULT NULL ,
     QQKS CHAR(1) CCSID 37 DEFAULT NULL ,
     QQTOTR DECIMAL(15, 0) DEFAULT NULL ,
     QQTMPR DECIMAL(15, 0) DEFAULT NULL ,
     QQJNP DECIMAL(15, 0) DEFAULT NULL ,
     QQEPT DECIMAL(15, 0) DEFAULT NULL ,
     QQDSS CHAR(1) CCSID 37 DEFAULT NULL ,
     QQIDXA CHAR(1) CCSID 37 DEFAULT NULL ,
     QQORDG CHAR(1) CCSID 37 DEFAULT NULL ,
     QQGRPG CHAR(1) CCSID 37 DEFAULT NULL ,
     QQJNG CHAR(1) CCSID 37 DEFAULT NULL ,
     QQUNIN CHAR(1) CCSID 37 DEFAULT NULL ,
     QQSUBQ CHAR(1) CCSID 37 DEFAULT NULL ,
     QQHSTV CHAR(1) CCSID 37 DEFAULT NULL ,
     QQRCDS CHAR(1) CCSID 37 DEFAULT NULL ,
     QQRCOD CHAR(2) CCSID 37 DEFAULT NULL ,
     QQRSS DECIMAL(15, 0) DEFAULT NULL ,
     QQREST DECIMAL(15, 0) DEFAULT NULL ,
     QQRIDX DECIMAL(15, 0) DEFAULT NULL ,
     QQFKEY DECIMAL(15, 0) DEFAULT NULL ,
     QQKSEL DECIMAL(15, 0) DEFAULT NULL ,
     QQAJN DECIMAL(15, 0) DEFAULT NULL ,
     QQIDXD VARCHAR(1000) ALLOCATE(48) CCSID 37 DEFAULT NULL ,
     QQC11 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC12 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC13 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC14 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC15 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC16 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC18 CHAR(1) CCSID 37 DEFAULT NULL ,
     QQC21 CHAR(2) CCSID 37 DEFAULT NULL ,
     QQC22 CHAR(2) CCSID 37 DEFAULT NULL ,
     QQC23 CHAR(2) CCSID 37 DEFAULT NULL ,
     QQI1 DECIMAL(15, 0) DEFAULT NULL ,
     QQI2 DECIMAL(15, 0) DEFAULT NULL ,
     QQI3 DECIMAL(15, 0) DEFAULT NULL ,
     QQI4 DECIMAL(15, 0) DEFAULT NULL ,
     QQI5 DECIMAL(15, 0) DEFAULT NULL ,
     QQI6 DECIMAL(15, 0) DEFAULT NULL ,
     QQI7 DECIMAL(15, 0) DEFAULT NULL ,
     QQI8 DECIMAL(15, 0) DEFAULT NULL ,
     QQI9 DECIMAL(15, 0) DEFAULT NULL ,
     QQIA DECIMAL(15, 0) DEFAULT NULL ,
     QQF1 DECIMAL(15, 0) DEFAULT NULL ,
     QQF2 DECIMAL(15, 0) DEFAULT NULL ,
     QQF3 DECIMAL(15, 0) DEFAULT NULL ,
     QQC61 CHAR(6) CCSID 37 DEFAULT NULL ,
     QQC81 CHAR(8) CCSID 37 DEFAULT NULL ,
     QQC82 CHAR(8) CCSID 37 DEFAULT NULL ,
     QQC83 CHAR(8) CCSID 37 DEFAULT NULL ,
     QQC84 CHAR(8) CCSID 37 DEFAULT NULL ,
     QQC101 CHAR(10) CCSID 37 DEFAULT NULL ,
     QQC102 CHAR(10) CCSID 37 DEFAULT NULL ,
     QQC103 CHAR(10) CCSID 37 DEFAULT NULL ,
     QQC104 CHAR(10) CCSID 37 DEFAULT NULL ,
     QQC105 CHAR(10) CCSID 37 DEFAULT NULL ,
     QQC106 CHAR(10) CCSID 37 DEFAULT NULL ,
     QQC181 VARCHAR(128) ALLOCATE(18) CCSID 37 DEFAULT NULL ,
```

```
    QQC182 VARCHAR(128) ALLOCATE(18) CCSID 37 DEFAULT NULL ,
    QQC183 VARCHAR(128) ALLOCATE(15) CCSID 37 DEFAULT NULL ,
    QQC301 VARCHAR(30) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QQC302 VARCHAR(30) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QQC303 VARCHAR(30) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QQ1000 VARCHAR(1000) ALLOCATE(48) CCSID 37 DEFAULT NULL ,
    QQTIM1 TIMESTAMP DEFAULT NULL ,
    QQTIM2 TIMESTAMP DEFAULT NULL ,
    QVQTBL VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QVQLIB VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QVPTBL VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QVPLIB VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QVINAM VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QVILIB VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
    QVQTBLI CHAR(1) CCSID 37 DEFAULT NULL ,
    QVPTBLI CHAR(1) CCSID 37 DEFAULT NULL ,
    QVINAMI CHAR(1) CCSID 37 DEFAULT NULL ,
    QVBNDY CHAR(1) CCSID 37 DEFAULT NULL ,
    QVJFANO CHAR(1) CCSID 37 DEFAULT NULL ,
    QVPARPF CHAR(1) CCSID 37 DEFAULT NULL ,
    QVPARPL CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC11 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC12 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC13 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC14 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC15 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC16 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC17 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC18 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC19 CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC1A CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC1B CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC1C CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC1D CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC1E CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC1F CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC11 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC12 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC13 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC14 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC15 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC16 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC17 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC18 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC19 CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC1A CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC1B CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC1C CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC1D CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC1E CHAR(1) CCSID 37 DEFAULT NULL ,
    QWC1F CHAR(1) CCSID 37 DEFAULT NULL ,
    QVC21 CHAR(2) CCSID 37 DEFAULT NULL ,
    QVC22 CHAR(2) CCSID 37 DEFAULT NULL ,
    QVC23 CHAR(2) CCSID 37 DEFAULT NULL ,
    QVC24 CHAR(2) CCSID 37 DEFAULT NULL ,
    QVCTIM DECIMAL(15, 0) DEFAULT NULL ,
    QVPARD DECIMAL(15, 0) DEFAULT NULL ,
    QVPARU DECIMAL(15, 0) DEFAULT NULL ,
    QVPARRC DECIMAL(15, 0) DEFAULT NULL ,
    QVRCNT DECIMAL(15, 0) DEFAULT NULL ,
    QVFILES DECIMAL(15, 0) DEFAULT NULL ,
    QVP151 DECIMAL(15, 0) DEFAULT NULL ,
    QVP152 DECIMAL(15, 0) DEFAULT NULL ,
    QVP153 DECIMAL(15, 0) DEFAULT NULL ,
    QVP154 DECIMAL(15, 0) DEFAULT NULL ,
    QVP155 DECIMAL(15, 0) DEFAULT NULL ,
    QVP156 DECIMAL(15, 0) DEFAULT NULL ,
    QVP157 DECIMAL(15, 0) DEFAULT NULL ,
    QVP158 DECIMAL(15, 0) DEFAULT NULL ,
    QVP159 DECIMAL(15, 0) DEFAULT NULL ,
    QVP15A DECIMAL(15, 0) DEFAULT NULL ,
    QVP15B DECIMAL(15, 0) DEFAULT NULL ,
    QVP15C DECIMAL(15, 0) DEFAULT NULL ,
    QVP15D DECIMAL(15, 0) DEFAULT NULL ,
    QVP15E DECIMAL(15, 0) DEFAULT NULL ,
    QVP15F DECIMAL(15, 0) DEFAULT NULL ,
    QVC41 CHAR(4) CCSID 37 DEFAULT NULL ,
    QVC42 CHAR(4) CCSID 37 DEFAULT NULL ,
    QVC43 CHAR(4) CCSID 37 DEFAULT NULL ,
    QVC44 CHAR(4) CCSID 37 DEFAULT NULL ,
    QVC81 CHAR(8) CCSID 37 DEFAULT NULL ,
    QVC82 CHAR(8) CCSID 37 DEFAULT NULL ,
```

```
      QVC83 CHAR(8) CCSID 37 DEFAULT NULL ,
      QVC84 CHAR(8) CCSID 37 DEFAULT NULL ,
      QVC85 CHAR(8) CCSID 37 DEFAULT NULL ,
      QVC86 CHAR(8) CCSID 37 DEFAULT NULL ,
      QVC87 CHAR(8) CCSID 37 DEFAULT NULL ,
      QVC88 CHAR(8) CCSID 37 DEFAULT NULL ,
      QVC101 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC102 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC103 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC104 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC105 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC106 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC107 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC108 CHAR(10) CCSID 37 DEFAULT NULL ,
      QVC1281 VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
      QVC1282 VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
      QVC1283 VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
      QVC1284 VARCHAR(128) ALLOCATE(10) CCSID 37 DEFAULT NULL ,
      QVC3001 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3002 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3003 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3004 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3005 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3006 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3007 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC3008 VARCHAR(300) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC5001 VARCHAR(500) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC5002 VARCHAR(500) ALLOCATE(32) CCSID 37 DEFAULT NULL ,
      QVC1000 VARCHAR(1000) ALLOCATE(48) CCSID 37 DEFAULT NULL ,
      QWC1000 VARCHAR(1000) ALLOCATE(48) CCSID 37 DEFAULT NULL ,
      QQINT01 INTEGER DEFAULT NULL ,
      QQINT02 INTEGER DEFAULT NULL ,
      QQINT03 INTEGER DEFAULT NULL ,
      QQINT04 INTEGER DEFAULT NULL ,
      QQSMINT1 SMALLINT DEFAULT NULL ,
      QQSMINT2 SMALLINT DEFAULT NULL ,
      QQSMINT3 SMALLINT DEFAULT NULL ,
      QQSMINT4 SMALLINT DEFAULT NULL ,
      QQSMINT5 SMALLINT DEFAULT NULL ,
      QQSMINT6 SMALLINT DEFAULT NULL ,
      QQ1000L CLOB(2147483647) ALLOCATE(48) CCSID 37 DEFAULT NULL ,
      QFC11 CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC12 CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC13 CHAR(1) CCSID 37 DEFAULT NULL ,
      QQCLOB2 CLOB(2147483647) ALLOCATE(48) CCSID 37 DEFAULT NULL ,
      QFC14 CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC15 CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC16 CHAR(1) CCSID 37 DEFAULT NULL ,
      QQCLOB3 CLOB(2147483647) CCSID 37 DEFAULT NULL ,
      QFC17 CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC18 CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC19 CHAR(1) CCSID 37 DEFAULT NULL ,
      QQDBCLOB1 DBCLOB(1073741823) ALLOCATE(24) CCSID 1200 DEFAULT NULL ,
      QFC1A CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC1B CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC1C CHAR(1) CCSID 37 DEFAULT NULL ,
      QQDBCLOB2 DBCLOB(1073741823) CCSID 1200 DEFAULT NULL ,
      QFC1D CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC1E CHAR(1) CCSID 37 DEFAULT NULL ,
      QFC1F CHAR(1) CCSID 37 DEFAULT NULL ,
      QQBLOB1 BLOB(2147483647) DEFAULT NULL ,
      QXC11 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC12 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC13 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC14 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC15 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC16 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC17 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC18 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC19 CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC1A CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC1B CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC1C CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC1D CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC1E CHAR(1) CCSID 37 DEFAULT NULL ,
      QXC21 CHAR(2) CCSID 37 DEFAULT NULL ,
      QXC22 CHAR(2) CCSID 37 DEFAULT NULL ,
      QXC23 CHAR(2) CCSID 37 DEFAULT NULL ,
      QXC24 CHAR(2) CCSID 37 DEFAULT NULL ,
      QXC25 CHAR(2) CCSID 37 DEFAULT NULL ,
      QXC26 CHAR(2) CCSID 37 DEFAULT NULL ,
      QXC27 CHAR(2) CCSID 37 DEFAULT NULL ,
```

```
     QXC28 CHAR(2) CCSID 37 DEFAULT NULL ,
     QXC29 CHAR(2) CCSID 37 DEFAULT NULL ,
     QXC41 CHAR(4) CCSID 37 DEFAULT NULL ,
     QXC42 CHAR(4) CCSID 37 DEFAULT NULL ,
     QXC43 CHAR(4) CCSID 65535 DEFAULT NULL ,
     QXC44 CHAR(4) CCSID 37 DEFAULT NULL ,
     QQINT05 INTEGER DEFAULT NULL ,
     QQINT06 INTEGER DEFAULT NULL ,
     QQINT07 INTEGER DEFAULT NULL ,
     QQINT08 INTEGER DEFAULT NULL ,
     QQINT09 INTEGER DEFAULT NULL ,
     QQINT0A INTEGER DEFAULT NULL ,
     QQINT0B INTEGER DEFAULT NULL ,
     QQINT0C INTEGER DEFAULT NULL ,
     QQINT0D INTEGER DEFAULT NULL ,
     QQINT0E INTEGER DEFAULT NULL ,
     QQINT0F INTEGER DEFAULT NULL ,
     QQSMINT7 SMALLINT DEFAULT NULL ,
     QQSMINT8 SMALLINT DEFAULT NULL ,
     QQSMINT9 SMALLINT DEFAULT NULL ,
     QQSMINTA SMALLINT DEFAULT NULL ,
     QQSMINTB SMALLINT DEFAULT NULL ,
     QQSMINTC SMALLINT DEFAULT NULL ,
     QQSMINTD SMALLINT DEFAULT NULL ,
     QQSMINTE SMALLINT DEFAULT NULL ,
     QQSMINTF SMALLINT DEFAULT NULL )

     RCDFMT QQQDBMN    ;
   RENAME QSYS/QQQDBMN TO SYSTEM NAME QAQQDBMN;

   LABEL ON TABLE QSYS/QAQQDBMN
     IS 'Database Monitor Physical File' ;

   LABEL ON COLUMN QSYS.QAQQDBMN
 ( QQRID IS 'Record              ID' ,
     QQTIME IS 'Created            Time' ,
     QQJFLD IS 'Join               Column' ,
     QQRDBN IS 'Relational         Database            Name' ,
     QQSYS IS 'System             Name' ,
     QQJOB IS 'Job                Name' ,
     QQUSER IS 'Job                User' ,
     QQJNUM IS 'Job                Number' ,
     QQUCNT IS 'Unique             Counter' ,
     QQUDEF IS 'User               Defined             Column' ,
     QQSTN IS 'Statement          Number' ,
     QQQDTN IS 'Subselect          Number' ,
     QQQDTL IS 'Subselect          Nested              Level' ,
     QQMATN IS 'Subselect          Number of           Materialized View' ,
     QQMATL IS 'Subselect          Level of            Materialized View' ,
     QQTLN IS 'Library of         Table               Queried' ,
     QQTFN IS 'Name of            Table               Queried' ,
     QQTMN IS 'Member of          Table               Queried' ,
     QQPTLN IS 'Library of         Base                Table' ,
     QQPTFN IS 'Name of            Base                Table' ,
     QQPTMN IS 'Member of          Base                Table' ,
     QQILNM IS 'Library of         Index               Used' ,
     QQIFNM IS 'Name of            Index               Used' ,
     QQIMNM IS 'Member of          Index               Used' ,
     QQNTNM IS 'NLSS               Table' ,
     QQNLNM IS 'NLSS               Library' ,
     QQSTIM IS 'Start              Time' ,
     QQETIM IS 'End                Time' ,
     QQKP IS 'Key                Positioning' ,
     QQKS IS 'Key                Selection' ,
     QQTOTR IS 'Total              Rows' ,
     QQTMPR IS 'Number             of Rows             in Temporary' ,
     QQJNP IS 'Join               Position' ,
     QQEPT IS 'Estimated          Processing          Time' ,
     QQDSS IS 'Data               Space               Selection' ,
     QQIDXA IS 'Index              Advised' ,
     QQORDG IS 'Ordering' ,
     QQGRPG IS 'Grouping' ,
     QQJNG IS 'Join' ,
     QQUNIN IS 'Union' ,
     QQSUBQ IS 'Subquery' ,
     QQHSTV IS 'Host               Variables' ,
     QQRCDS IS 'Row                Selection' ,
     QQRCOD IS 'Reason             Code' ,
     QQRSS IS 'Number             of Rows             Selected' ,
     QQREST IS 'Estimated          Number of           Rows Selected' ,
     QQRIDX IS 'Number of          Entries in          Index Created' ,
     QQFKEY IS 'Estimated          Entries for         Key Positioning' ,
```

```
    QQKSEL IS 'Estimated           Entries for         Key Selection' ,
    QQAJN IS 'Estimated           Number of           Joined Rows' ,
    QQIDXD IS 'Advised             Key                 Columns' ,
    QQI9 IS 'Thread              Identifier' ,
    QVQTBL IS 'Queried             Table               Long Name' ,
    QVQLIB IS 'Queried             Library             Long Name' ,
    QVPTBL IS 'Base                Table               Long Name' ,
    QVPLIB IS 'Base                Library             Long Name' ,
    QVINAM IS 'Index Used          Long Name' ,
    QVILIB IS 'Index Used          Library             Name' ,
    QVQTBLI IS 'Table               Long                Required' ,
    QVPTBLI IS 'Base                Long                Required' ,
    QVINAMI IS 'Index               Long                Required' ,
    QVBNDY IS 'I/O or CPU          Bound' ,
    QVJFANO IS 'Join                Fan                 Out' ,
    QVPARPF IS 'Parallel            Pre-Fetch' ,
    QVPARPL IS 'Parallel            Pre-Load' ,
    QVCTIM IS 'Estimated           Cumulative          Time' ,
    QVPARD IS 'Parallel            Degree              Requested' ,
    QVPARU IS 'Parallel            Degree              Used' ,
    QVPARRC IS 'Parallel            Limited             Reason Code' ,
    QVRCNT IS 'Refresh             Count' ,
    QVFILES IS 'Number of           Tables              Joined' ) ;

LABEL ON COLUMN QSYS.QAQQDBMN
( QQRID TEXT IS 'Record ID' ,
    QQTIME TEXT IS 'Time record was created' ,
    QQJFLD TEXT IS 'Join Column' ,
    QQRDBN TEXT IS 'Relational Database Name' ,
    QQSYS TEXT IS 'System Name' ,
    QQJOB TEXT IS 'Job Name' ,
    QQUSER TEXT IS 'Job User' ,
    QQJNUM TEXT IS 'Job Number' ,
    QQUCNT TEXT IS 'Unique Counter' ,
    QQUDEF TEXT IS 'User Defined Column' ,
    QQSTN TEXT IS 'Statement Number' ,
    QQQDTN TEXT IS 'Subselect Number' ,
    QQQDTL TEXT IS 'Subselect Nested Level' ,
    QQMATN TEXT IS 'Subselect Number of Materialized View' ,
    QQMATL TEXT IS 'Subselect Level of Materialized View' ,
    QQTLN TEXT IS 'Library of Table Queried' ,
    QQTFN TEXT IS 'Name of Table Queried' ,
    QQTMN TEXT IS 'Member of Table Queried' ,
    QQPTLN TEXT IS 'Base Table Library' ,
    QQPTFN TEXT IS 'Base Table' ,
    QQPTMN TEXT IS 'Base Table Member' ,
    QQILNM TEXT IS 'Library of Index Used' ,
    QQIFNM TEXT IS 'Name of Index Used' ,
    QQIMNM TEXT IS 'Member of Index Used' ,
    QQNTNM TEXT IS 'NLSS Table' ,
    QQNLNM TEXT IS 'NLSS Library' ,
    QQSTIM TEXT IS 'Start timestamp' ,
    QQETIM TEXT IS 'End timestamp' ,
    QQKP TEXT IS 'Key positioning' ,
    QQKS TEXT IS 'Key selection' ,
    QQTOTR TEXT IS 'Total row in table' ,
    QQTMPR TEXT IS 'Number of rows in temporary' ,
    QQJNP TEXT IS 'Join Position' ,
    QQEPT TEXT IS 'Estimated processing time' ,
    QQDSS TEXT IS 'Data Space Selection' ,
    QQIDXA TEXT IS 'Index advised' ,
    QQORDG TEXT IS 'Ordering' ,
    QQGRPG TEXT IS 'Grouping' ,
    QQJNG TEXT IS 'Join' ,
    QQUNIN TEXT IS 'Union' ,
    QQSUBQ TEXT IS 'Subquery' ,
    QQHSTV TEXT IS 'Host Variables' ,
    QQRCDS TEXT IS 'Row Selection' ,
    QQRCOD TEXT IS 'Reason Code' ,
    QQRSS TEXT IS 'Number of rows selected or sorted' ,
    QQREST TEXT IS 'Estimated number of rows selected' ,
    QQRIDX TEXT IS 'Number of entries in index created' ,
    QQFKEY TEXT IS 'Estimated keys for key positioning' ,
    QQKSEL TEXT IS 'Estimated keys for key selection' ,
    QQAJN TEXT IS 'Estimated number of joined rows' ,
    QQIDXD TEXT IS 'Key columns for the index advised' ,
    QQI9 TEXT IS 'Thread Identifier' ,
    QVQTBL TEXT IS 'Queried Table, Long Name' ,
    QVQLIB TEXT IS 'Queried Library, Long Name' ,
    QVPTBL TEXT IS 'Base Table, Long Name' ,
    QVPLIB TEXT IS 'Base Library, Long Name' ,
    QVINAM TEXT IS 'Index Used, Long Name' ,
```

```
    QVILIB TEXT IS 'Index Used, Libary Name' ,
    QVQTBLI TEXT IS 'Table Long Required' ,
    QVPTBLI TEXT IS 'Base Long Required' ,
    QVINAMI TEXT IS 'Index Long Required' ,
    QVBNDY TEXT IS 'I/O or CPU Bound' ,
    QVJFANO TEXT IS 'Join Fan out' ,
    QVPARPF TEXT IS 'Parallel Pre-Fetch' ,
    QVPARPL TEXT IS 'Parallel Pre-Load' ,
    QVCTIM TEXT IS 'Cumulative Time' ,
    QVPARD TEXT IS 'Parallel Degree, Requested' ,
    QVPARU TEXT IS 'Parallel Degree, Used' ,
    QVPARRC TEXT IS 'Parallel Limited, Reason Code' ,
    QVRCNT TEXT IS 'Refresh Count' ,
    QVFILES TEXT IS 'Number of, Tables Joined' ) ;
```

Optional database monitor SQL view format

These examples show the different optional SQL view format that you can create with the SQL shown. The column descriptions are explained in the tables following each example. These views are not shipped with the system, and you must create them, if you choose to do so. These views are optional and are not required for analyzing monitor data.

Any rows that have a row identification number (QQRID) of 5000 or greater are for internal database use.

Database monitor view 1000 - SQL Information

Displays the SQL logical view format for database monitor QQQ1000.

```
Create View QQQ1000 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQI5 as Unique_Refresh_Counter,
          QQUDEF as User_Defined,
          QQSTN as Statement_Number,
          QQC11 as Statement_Function,
          QQC21 as Statement_Operation,
          QQC12 as Statement_Type,
          QQC13 as Parse_Required,
          QQC103 as Package_Name,
          QQC104 as Package_Library,
          QQC181 as Cursor_Name,
          QQC182 as Statement_Name,
          QQSTIM as Start_Timestamp,
          QQ1000 as Statement_Text,
          QQC14 as Statement_Outcome,
          QQI2 as Result_Rows,
          QQC22 as Dynamic_Replan_Reason_Code,
          QQC16 as Data_Conversion_Reason_Code,
          QQI4 as Total_Time_Milliseconds,
          QQI3 as Rows_Fetched,
          QQETIM as End_Timestamp,
          QQI6 as Total_Time_Microseconds,
          QQI7 as SQL_Statement_Length,
          QQI1 as Insert_Unique_Count,
          QQI8 as SQLCode,
          QQC81 as SQLState,
          QVC101 as Close_Cursor_Mode,
          QVC11 as Allow_Copy_Data_Value,
          QVC12 as PseudoOpen,
          QVC13 as PseudoClose,
          QVC14 as ODP_Implementation,
          QVC21 as Dynamic_Replan_SubCode,
          QVC41 as Commitment_Control_Level,
          QWC1B as Concurrent_Access_Resolution,
          QVC15 as Blocking_Type,
          QVC16 as Delay_Prepare,
          QVC1C as Explainable,
          QVC17 as Naming_Convention,
          QVC18 as Dynamic_Processing_Type,
```

```
            QVC19 as LOB_Data_Optimized,
            QVC1A as Program_User_Profile_Used,
            QVC1B as Dynamic_User_Profile_Used,
            QVC1281 as Default_Collection,
            QVC1282 as Procedure_Name,
            QVC1283 as Procedure_Library,
            QQCLOB2 as SQL_Path,
            QVC1284 as Current_Schema,
            QQC18 as Binding_Type,
            QQC61 as Cursor_Type,
            QVC1D as Statement_Originator,
            QQC15 as Hard_Close_Reason_Code,
            QQC23 as Hard_Close_Subcode,
            QVC42 as Date_Format,
            QWC11 as Date_Separator,
            QVC43 as Time_Format,
            QWC12 as Time_Separator,
            QWC13 as Decimal_Point,
            QVC104 as Sort_Sequence_Table ,
            QVC105 as Sort_Sequence_Library,
            QVC44 as Language_ID,
            QVC23 as Country_ID,
            QQIA as First_N_Rows_Value,
            QQF1 as Optimize_For_N_Rows_Value,
            QVC22 as SQL_Access_Plan_Reason_Code,
            QVC24 as Access_Plan_Not_Saved_Reason_Code,
            QVC81 as Transaction_Context_ID,
            QVP152 as Activation_Group_Mark,
            QVP153 as Open_Cursor_Threshold,
            QVP154 as Open_Cursor_Close_Count,
            QVP155 as Commitment_Control_Lock_Limit,
            QWC15 as Allow_SQL_Mixed_Constants,
            QWC16 as Suppress_SQL_Warnings,
            QWC17 as Translate_ASCII,
            QWC18 as System_Wide_Statement_Cache,
            QVP159 as LOB_Locator_Threshold,
            QVP156 as Max_Decimal_Precision,
            QVP157 as Max_Decimal_Scale,
            QVP158 as Min_Decimal_Divide_Scale ,
            QWC19 as Unicode_Normalization,
            QQ1000L as Statement_Text_Long,
            QVP15B as Old_Access_Plan_Length,
            QVP15C as New_Access_Plan_Length,
            QVP151 as Fast_Delete_Count,
            QQF2 as Statement_Max_Compression,
            QVC102 as Current_User_Profile,
            QVC1E as Expression_Evaluator_Used,
            QVP15A as Host_Server_Delta,
            QQC301 as NTS_Lock_Space_Id,
            QQC183 as IP_Address,
            QFC11 as IP_Type,
            QQSMINT2 as IP_Port_Number,
            QVC3004 as NTS_Transaction_Id,
            QQSMINT3 as NTS_Format_Id_Length,
            QQSMINT4 as NTS_Transatction_ID_SubLength,
            QVRCNT as Unique_Refresh_Counter2,
            QVP15F as Times_Run,
            QVP15E as FullOpens,
            QVC1F as Proc_In_Cache,
            QWC1A as Combined_Operation,
            QVC3001 as Client_Applname,
            QVC3002 as Client_Userid,
            QVC3003 as Client_Wrkstnname,
            QVC3005 as Client_Acctng,
            QVC3006 as Client_Progamid,
            QVC5001 as Interface_Information,
            QVC82 as Open_Options,
            QWC1D as Extended_Indicators,
            QWC1C as DECFLOAT_Rounding_Mode,
            QWC1E as SQL_DECFLOAT_Warnings,
            QVP15D as Worst_Time_Micro,
            QQINT05 as SQ_Unique_Count,
            QFC13 as Concurrent_Access_Res_Used,
            QQSMINT8 as SQL_Scalar_UDFs_Not_Inlined,
            QVC3007 as Result_Set_Cursor,
            QFC12 as Implicit_XMLPARSE_Option,
            QQSMINT7 as SQL_XML_Data_CCSID,
            QQSMINT5 as OPTIMIZER_USE,
            QFC14 as XML_Schema_In_Cache,
            QQC105 as Current_User,
            QFC15 as Row_Column_Access_Control
```

```
FROM    DbMonLib/DbMonTable
WHERE   QQRID=1000)
```

Table 130. QQQ1000 - SQL Information

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
Unique_Refresh_Counter	QQI5	Unique refresh counter
User_Defined	QQUDEF	User-defined column
Statement_Number	QQSTN	Statement number (unique per statement)
Statement_Function	QQC11	Statement function: • S - Select • U - Update • I - Insert • D - Delete • L - Data definition language • O - Other

Table 130. QQQ1000 - SQL Information (continued)		
View Column Name	**Table Column Name**	**Description**
Statement_Operation	QQC21	Statement operation: • AC - Allocate cursor • AD - Allocate descriptor • AF - Alter function • AL - Alter table • AK - Alter mask • AP - Alter procedure • AQ - Alter sequence • AR - Alter permission • AS - Associate locators • AT - Alter trigger • BE - Compound (dynamic) • CA - Call • CB - Create variable • CC - Create collection • CD - Create type • CF - Create function • CG - Create trigger • CI - Create index • CK - Create mask • CL - Close • CM - Commit • CN - Connect • CO - Comment on • CP - Create procedure • CQ - Create sequence • CR - Create permission • CS - Create alias/synonym • CT - Create table • CV - Create view • DA - Deallocate descriptor • DE - Describe • DI - Disconnect • DL - Delete • DM - Describe parameter marker • DO - Describe procedure • DP - Declare procedure • DR - Drop • DS - Describe cursor • DT - Describe table • EI - Execute immediate • EX - Execute • FE - Fetch • FL - Free locator • GR - Grant • GS - Get descriptor • HC - Hard close • HL - Hold locator • IN - Insert

View Column Name	Table Column Name	Description
Statement_Operation (continued)	QQC21	• JR - Server job reused • LK - Lock • LO - Label on • MG - Merge • MT - More text (Deprecated in V5R4) • OP - Open • PD - Prepare and describe • PR - Prepare • QF - OPNQRYF command • QM - Query/400 STRQMQRY command • QO - OPNDBF command or Native open • QQ - QQQQRY() API • QR - RUNQRY command • RB - Rollback to savepoint • RE - Release • RF - Refresh Table • RG - Resignal • RM - Set current DECFLOAT rounding mode • RO - Rollback • RS - Release Savepoint • RT - Rename table • RV - Revoke • SA - Savepoint • SC - Set connection • SD - Set descriptor • SE - Set encryption password • SN - Set session user • SI - Select into • SO - Set current degree • SP - Set path • SR - Set result set • SS - Set current schema • ST - Set transaction • SV - Set variable • SX - Set current implicit XMLPARSE option • TO - Transfer ownership • TT - Truncate • UP - Update • VI - Values into • X0 - Unknown statement • X1 - Unknown statement • X2 - DRDA (AS) Unknown statement • X3 - Unknown statement • X9 - Internal error • XA - X/Open API • ZD - Host server only activity

Table 130. QQQ1000 - SQL Information (continued)

Table 130. QQQ1000 - SQL Information (continued)		
View Column Name	**Table Column Name**	**Description**
Statement_Type	QQC12	Statement type: • D - Dynamic statement • S - Static statement
Parse_Required	QQC13	Parse required (Y/N)
Package_Name	QQC103	Name of the package or name of the program that contains the current SQL statement
Package_Library	QQC104	Name of the library containing the package
Cursor_Name	QQC181	Name of the cursor corresponding to this SQL statement, if applicable
Statement_Name	QQC182	Name of statement for SQL statement, if applicable
Start_Timestamp	QQSTIM	Time this statement entered
Statement_Text	QQ1000	First 1000 bytes of statement text
Statement_Outcome	QQC14	Statement outcome • S - Successful • U - Unsuccessful
Result_Rows	QQI2	Number of result rows returned. Will only be set for the following SQL operations and is 0 for all others: • IN - Insert • UP - Update • DL - Delete • For an SQL Plan Cache snapshot, this count represents the aggregate count for all runs of this query. This count can be divided by the total number of runs, COALESCE(QVP15F,1), to determine the average rows fetched for a given query run.

View Column Name	Table Column Name	Description
Dynamic_Replan_Reason_Code	QQC22	Dynamic replan (access plan rebuilt) • NA - No replan. • NR - SQL QDT rebuilt for new release. • A1 - A table or member is not the same object as the one referenced when the access plan was last built. Some reasons why they might be different are: – Object was deleted and recreated. – Object was saved and restored. – Library list was changed. – Object was renamed. – Object was moved. – Object was overridden to a different object. – This run is the first run of this query after the object containing the query has been restored. – Mask or permission attributes changed for the object. • A2 - Access plan was built to use a reusable Open Data Path (ODP) and the optimizer chose to use a nonreusable ODP for this call. • A3 - Access plan was built to use a non-reusable Open Data Path (ODP) and the optimizer chose to use a reusable ODP for this call. • A4 - Either the number of rows in the table member has changed by more than 10% or a selectivity or cardinality statistic has change by more than 25% since the access plan was last built. • A5 - A new index exists over one of the tables in the query. • A6 - An index that was used for this access plan no longer exists or is no longer valid. • A7 - IBM i Query requires the access plan to be rebuilt because of system programming changes. • A8 - The CCSID of the current job is different from the CCSID of the job that last created the access plan. • A9 - The value of one or more of the following values is different for the current job than it was for the job that last created this access plan: – date format – date separator – time format – time separator

Table 130. QQQ1000 - SQL Information (continued)

Table 130. QQQ1000 - SQL Information (continued)		
View Column Name	**Table Column Name**	**Description**
Dynamic_Replan_Reason_Code (continued)	QQC22	• AA - The sort sequence table specified is different from the sort sequence table that was used when this access plan was created. • AB - Storage pool changed or DEGREE parameter of CHGQRYA command changed. • AC - The system feature DB2 Symmetric Multiprocesing has been installed or removed. • AD - The value of the degree query attribute has changed. • AE - A view is either being opened by a high-level language or a view is being materialized. • AF - A user-defined type or user-defined function is not the same object as the one referred to in the access plan; or the SQL Path is not the same as when the access plan was built. • B0 - The options specified have changed as a result of the query options file. • B1 - The access plan was generated with a commitment control level that is different in the current job. • B2 - The access plan was generated with a static cursor answer set size that is different from the previous access plan. • B3 - The query was reoptimized because this run is the first run of the query after it was prepared. This run is the first run with actual parameter marker values. • B4 - The query was reoptimized because referential or check constraints have changed. • B5 - The query was reoptimized because Materialized query tables have changed. • B6 - The query was reoptimized because the value of a host variable changed and the access plan is no longer valid. • B7 - The query was reoptimized because AQP determined that it was beneficial. • B8 - The query was reoptimized because Expression Evaluator determined that the statement should be reoptimized.
Data_Conversion_Reason_Code	QQC16	Data conversion • N - No. • 0 - Not applicable. • 1 - Lengths do not match. • 2 - Numeric types do not match. • 3 - C host variable is NUL-terminated. • 4 - Host variable or column is variable length and the other is not variable length. • 5 - Host variable or column is not variable length and the other is variable length. • 6 - Host variable or column is variable length and the other is not variable length. • 7 - CCSID conversion. • 8 - DRDA and NULL capable, variable length, contained in a partial row, derived expression, or blocked fetch with not enough host variables. • 9 - Target table of an insert is not an SQL table.

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
Data_Conversion_Reason_Code (continued)		• 10 - Host variable is too short to hold a TIME or TIMESTAMP value being retrieved. • 11 - Host variable is DATE, TIME, or TIMESTAMP and value being retrieved is a character string. • 12 - Too many host variables specified and records are blocked. • 13 - DRDA used for a blocked FETCH. Also, the number of host variables specified in the INTO clause is less than the number of result values in the select list. • 14 - LOB locator used and the commitment control level was not *ALL.
Total_Time_Milliseconds	QQI4	Total time for this statement, in milliseconds. For fetches, the time includes all fetches for this OPEN of the cursor. Note: When monitor files are created when using an SQL Plan Cache snapshot, this time represents the aggregate time for all runs of this query. This time can be divided by the total number of runs, COALESCE(QVP15F,1), to determine an average time for a given run of the query.
Rows_Fetched	QQI3	Total rows fetched for cursor Note: When monitor files are created when using an SQL Plan Cache snapshot, this field is not set.
End_Timestamp	QQETIM	Time SQL request completed
Total_Time_Microseconds	QQI6	Total time for this statement, in microseconds. For fetches, this time includes all fetches for this OPEN of the cursor. Note: When monitor files are created when using an SQL Plan Cache snapshot, this time represents the aggregate time for all runs of this query. This time can be divided by the total number of runs, COALESCE(QVP15F,1), to determine an average time for a given run of the query.
SQL_Statement_Length	QQI7	Length of SQL Statement

Table 130. QQQ1000 - SQL Information (continued)		
View Column Name	**Table Column Name**	**Description**
Insert_Unique_Count	QQI1	If the operation (QQC21) indicates INSERT (IN), this field contains the unique query count for the QDT associated with the INSERT. QQUCNT contains the unique query count for the QDT associated with the WHERE part of the statement. If the operation (QQC21) indicates DELETE (DL) or TRUNCATE (TT), this field contains the fast delete reason code. Possible values if the operation is a DELETE or TRUNCATE are : • 0 - Fast delete results unknown or fast delete is not relevant because the delete failed. • 1 - Fast delete was achieved. All other values if the operation is a DELETE or TRUNCATE indicate the reason the database was unable to implement the request using fast delete. Fast delete attempt denied values: • 2 - File is a DDM file. • 3 - File is a multi member file. • 4 - File is distributed file. • 5 - File is a logical file or SQL view. • 6 - File is a parent file. • 7 - File has one or more enabled delete triggers created over it. • 8 - Number of rows in table is less than 1000 OR less than the QAQQINI SQL_FAST_DELETE_ROW_COUNT value. Refer to QVP151 to see the SQL_FAST_DELETE_ROW_COUNT value in effect for this statement. • 9 - DBMAINT failed. This reason code could appear for many reasons, including the existence of a logical open within this job, pending record changes, ragged save in progress and possibly other reasons. • 10- Failed to acquire an exclusive no read (LENR) lock on the file. • 11- Failed to acquire an exclusive allow read (LEAR) lock on the file's data space. • 12- The user does not have *EXECUTE authority to the library. • 13- File has one or more enabled delete triggers and RESTRICT WHEN DELETE TRIGGERS was specified on a TRUNCATE statement. • 51- A WHERE clause was used on the DELETE. • 52- QAQQINI SQL_FAST_DELETE_ROW_COUNT indicated to disallow fast delete. • 53- File is an alias referring to a partition table member. • 54- The user does not have *DELETE authority to the file. • 55- File is not found.
SQLCode	QQI8	SQL return code
SQLState	QQC81	SQLSTATE
Close_Cursor_Mode	QVC101	Close Cursor. Possible values are: • *ENDJOB - SQL cursors are closed when the job ends. • *ENDMOD - SQL cursors are closed when the module ends • *ENDPGM - SQL cursors are closed when the program ends. • *ENDSQL - SQL cursors are closed when the first SQL program on the call stack ends. • *ENDACTGRP - SQL cursors are closed when the activation group ends.

View Column Name	Table Column Name	Description
Allow_Copy_Data_Value	QVC11	ALWCPYDTA setting (Y/N/O) • Y - A copy of the data might be used. • N - Cannot use a copy of the data. • O - The optimizer can choose to use a copy of the data for performance.
PseudoOpen	QVC12	Pseudo Open (Y/N) for SQL operations that can trigger opens. • OP - Open • IN - Insert • UP - Update • DL - Delete • SI - Select Into • SV - Set • VI - Values into For all operations, it can be blank.
PseudoClose	QVC13	Pseudo Close (Y/N) for SQL operations that can trigger a close. • CL - Close • IN - Insert • UP - Update • DL - Delete • SI - Select Into • SV - Set • VI - Values into For all operations, it can be blank.
ODP_Implementation	QVC14	ODP implementation • R - Reusable ODP • N - Nonreusable ODP • ' ' - Column not used
Dynamic_Replan_SubCode	QVC21	Dynamic replan, subtype reason code
Commitment_Control_Level	QVC41	Commitment control level. Possible values are: • CS - Cursor stability • CSKL - Cursor stability. Keep exclusive locks. • NC - No commit • RR - Repeatable read • RREL - Repeatable read. Keep exclusive locks. • RS - Read stability • RSEL - Read stability. Keep exclusive locks. • UR - Uncommitted read
Concurrent_Access_Resolution	QWC1B	Indicates what method of concurrent access resolution was specified. • N - Concurrent access resolution was not specified. • S - SKIP LOCKED DATA clause was specified. • U - USE CURRENTLY COMMITTED clause was specified. • W- WAIT FOR OUTCOME clause was specified.

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
Blocking_Type	QVC15	Type of blocking. Possible values are: • S - Single row, ALWBLK(*READ) • F - Force one row, ALWBLK(*NONE) • L - Limited block, ALWBLK(*ALLREAD)
Delay_Prepare	QVC16	Delay prepare of statement (Y/N).
Explainable	QVC1C	The SQL statement is explainable (Y/N).
Naming_Convention	QVC17	Naming convention. Possible values: • N - System naming convention • S - SQL naming convention
Dynamic_Processing_Type	QVC18	Type of dynamic processing. • E - Extended dynamic • S - System wide cache • L - Local prepared statement
LOB_Data_Optimized	QVC19	Optimize LOB data types (Y/N)
Program_User_Profile_Used	QVC1A	User profile used when compiled programs are executed. Possible values are: • N = User Profile is determined by naming conventions. For *SQL, USRPRF(*OWNER) is used. For *SYS, USRPRF(*USER) is used. • U = USRPRF(*USER) is used. • O = USRPRF(*OWNER) is used.
Dynamic_User_Profile_Used	QVC1B	User profile used for dynamic SQL statements. • U = USRPRF(*USER) is used. • O = USRPRF(*OWNER) is used.
Default_Collection	QVC1281	Name of the default collection.
Procedure_Name	QVC1282	Procedure name on CALL to SQL.
Procedure_Library	QVC1283	Procedure library on CALL to SQL.
SQL_Path	QQCLOB2	Path used to find procedures, functions, and user-defined types for static SQL statements.
Current_Schema	QVC1284	SQL current schema.
Binding_Type	QQC18	Binding type: • C - Column-wise binding • R - Row-wise binding
Cursor_Type	QQC61	Cursor Type: • NSA - Non-scrollable, asensitive, forward only • NSI - Non-scrollable, insensitive, forward only • NSS - Non-scrollable, sensitive, forward only • SCA - scrollable, asensitive • SCI - scrollable, insensitive • SCS - scrollable, sensitive
Statement_Originator	QVC1D	SQL statement originator: • U - User • S - System

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
Hard_Close_Reason_Code	QQC15	SQL cursor hard close reason. Possible reasons are: • 1 - Internal Error • 2 - Exclusive Lock • 3 - Interactive SQL Reuse Restriction • 4 - Host variable Reuse Restriction • 5 - Temporary Result Restriction • 6 - Cursor Restriction • 7 - Cursor Hard Close Requested • 8 - Internal Error • 9 - Cursor Threshold • A - Optimizer decided to Hard-Close • B - Reuse Cursor Error • C - DRDA AS Cursor Closed • D - DRDA AR Not WITH HOLD • E - Repeatable Read • F - Lock Conflict Or QSQPRCED Threshold - Library • G - Lock Conflict Or QSQPRCED Threshold - File • H - Execute Immediate Access Plan Space • I - QSQCSRTH Dummy Cursor Threshold • J - File Override Change • K - Program Invocation Change • L - File Open Options Change • M - Statement Reuse Restriction • N - Internal Error • O - Library List Changed • P - Exit Processing • Q - SET SESSION USER statement
Hard_Close_Subcode	QQC23	SQL cursor hard close reason subcode. For QQC15 Reason code 'A' the following subcodes apply: • Z7 - New Index found • Z8 – Data Space Size changed out side of range • Z9 – MQT refresh age expired • ZA – Host variable values are no longer compatible with current plan • ZB – new statistic was found • ZC – commit level changed • ZD – Reoptimze for Warm IO • ZE – Reoptimze and change from FIRSTIO to ALLIO • ZF – Host variable selectivity changes require Reoptimization • ZG – AQP decided to hard-close

Table 130. QQQ1000 - SQL Information (continued)		
View Column Name	**Table Column Name**	**Description**
Date_Format	QVC42	Date Format. Possible values are: • ISO • USA • EUR • JIS • JUL • MDY • DMY • YMD
Date_Separator	QWC11	Date Separator. Possible values are: • "/" • "." • "," • "-" • " "
Time_Format	QVC43	Time Format. Possible values are: • ISO • USA • EUR • JIS • HMS
Time_Separator	QWC12	Time Separator. Possible values are: • ":" • "." • "," • " "
Decimal_Point	QWC13	Decimal Point. Possible values are: • "." • ","
Sort_Sequence_Table	QVC104	Sort Sequence Table
Sort_Sequence_Library	QVC105	Sort Sequence Library
Language_ID	QVC44	Language ID
Country_ID	QVC23	Country ID
First_N_Rows_Value	QQIA	Value specified on the FIRST n ROWS clause.
Optimize_For_N_Rows _Value	QQF1	Value specified on the OPTIMIZE FOR n ROWS clause.

View Column Name	Table Column Name	Description

SQL_Access_Plan_Reason_Code	QVC22	SQL access plan rebuild reason code. Possible reasons are:
		• A1 - A table or member is not the same object as the one referenced when the access plan was last built. Some reasons they might be different are:
		  – Object was deleted and recreated.
		  – Object was saved and restored.
		  – Library list was changed.
		  – Object was renamed.
		  – Object was moved.
		  – Object was overridden to a different object.
		  – This rebuild is the first run of this query after the object containing the query has been restored.
		• A2 - Access plan was built to use a reusable Open Data Path (ODP) and the optimizer chose to use a non-reusable ODP for this call.
		• A3 - Access plan was built to use a non-reusable Open Data Path (ODP) and the optimizer chose to use a reusable ODP for this call.
		• A4 - The number of rows in the table has changed by more than 10% since the access plan was last built.
		• A5 - A new index exists over one of the tables in the query
		• A6 - An index that was used for this access plan no longer exists or is no longer valid.

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
SQL_Access_Plan_Reason_Code (continued)		• A7 - IBM i Query requires the access plan to be rebuilt because of system programming changes. • A8 - The CCSID of the current job is different from the CCSID of the job that last created the access plan. • A9 - One or more of the following values is different for the current job than it was for the job that last created this access plan: – date format – date separator – time format – time separator. • AA - The sort sequence table specified is different from the sort sequence table that was used when this access plan was created. • AB - Storage pool changed or DEGREE parameter of CHGQRYA command changed. • AC - The system feature DB2 Symmetric Multiprocessing has been installed or removed. • AD - The value of the degree query attribute has changed. • AE- A view is either being opened by a high-level language or a view is being materialized. • AF - A user-defined type or user-defined function is not the same object as the one referred to in the access plan, or, the SQL Path is not the same as when the access plan was built. • B0 - The options specified have changed as a result of the query options file. • B1 - The access plan was generated with a commitment control level that is different in the current job. • B2 - The access plan was generated with a static cursor answer set size that is different from the previous access plan. • B3 - The query was reoptimized because this run is the first run after the query was prepared. It is the first run with actual parameter marker values. • B4 - The query was reoptimized because referential or check constraints have changed. • B5 - The query was reoptimized because Materialized query tables have changed. • B6 - The query was reoptimized because the value of a host variable changed and the access plan is no longer valid. • B7 - The query was reoptimized because AQP determined that the query must be reoptimized.

Table 130. QQQ1000 - SQL Information (continued)

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
Access_Plan_Not_Saved_Reason_Code	QVC24	Access plan not saved reason code. Possible reasons are: • A1 - Failed to get an LSUP lock on associated space of program or package. • A2 - Failed to get an immediate LEAR space location lock on first byte of associated space of program. • A3 - Failed to get an immediate LENR space location lock on first byte of associated space of program. • A5 - Failed to get an immediate LEAR space location lock on first byte of ILE associated space of a program. • A6 - Error trying to extend space of an ILE program. • A7 - No room in program. • A8 - No room in program associated space. • A9 - No room in program associated space. • AA - No need to save. Save already done in another job. • AB - Query optimizer cannot lock the QDT. • B1 - Saved at the end of the program associated space. • B2 - Saved at the end of the program associated space. • B3 - Saved in place. • B4 - Saved in place. • B5 - Saved at the end of the program associated space. • B6 - Saved in place. • B7 - Saved at the end of the program associated space. • B8 - Saved at the end of the program associated space.
Transaction_Context_ID	QVC81	Transaction context ID.
Activation_Group_Mark	QVP152	Activation Group Mark
Open_Cursor_Threshold	QVP153	Open cursor threshold
Open_Cursor_Close_Count	QVP154	Open cursor close count
Commitment_Control_Lock_Limit	QVP155	Commitment control lock limit
Allow_SQL_Mixed_Constants	QWC15	Using SQL mixed constants (Y/N)
Suppress_SQL_Warnings	QWC16	Suppress SQL warning messages (Y/N)
Translate_ASCII	QWC17	Translate ASCII to job (Y/N)
System_Wide_Statement_Cache	QWC18	Using system-wide SQL statement cache (Y/N)
LOB_Locator_Threshold	QVP159	LOB locator threshold
Max_Decimal_Precision	QVP156	Maximum decimal precision (63/31)
Max_Decimal_Scale	QVP157	Maximum decimal scale
Min_Decimal_Divide_Scale	QVP158	Minimum decimal divide scale
Unicode_Normalization	QWC19	Unicode data normalization requested (Y/N)
Statement_Text_Long	QQ1000L	Complete statement text
Old_Access_Plan_Length	QVP15B	Length of old access plan
New_Access_Plan_Length	QVP15C	Length of new access plan

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
Fast_Delete_Count	QVP151	SQL fast delete count. Possible values are: • 0 = *OPTIMIZE or *DEFAULT • 1-999,999,999,999 = User specified value • 'FFFFFFFFFFFFFFFF'x = *NONE
Statement_Max_Compression	QQF2	SQL statement maximum compression. Possible values are: • 1 - *DEFAULT • 1 - User specified queries • 2 - All queries, user, and system • 3 - System generated internal queries
Current_User_Profile	QVC102	Current user profile name
Expression_Evaluator_Used	QVC1E	• N - Not applicable • S - SQL mapping • Y - QQ expression evaluator • O - Expression handled by an Open • T - Expression evaluator used temporary copy of access plan
Host_Server_Delta	QVP15A	Time not spent within Host Server
NTS_Lock_Space_Id	QQC301	NTS Lock Space Identifier
IP_Address	QQC183	IP Address
IP_Type	QFC11	IP address type • '0' = No client IP address • '1' = IPV4 format • '2' = IPV6 format Only applicable for database server jobs.
IP_Port_Number	QQSMINT2	IP Port Number
NTS_Transaction_Id	QVC3004	NTS Transaction Identifier
NTS_Format_Id_Length	QQSMINT3	NTS Format Identified length
NTS_Transaction_ID_SubLength	QQSMINT4	NTS Transaction Identifier sublength.
Unique_Refresh_Counter2	QVRCNT	Unique refresh counter
Times_Run	QVP15F	Number of times this Statement was run. If Null, then the statement was run once. Note: While using an SQL Plan Cache snapshot, this value can be set by the database monitor. This value might be null if the query never completed, or was running when the snapshot was created. If there is not a plan cache snapshot, the value is null.
Full_Opens	QVP15E	Number of runs that were processed as full opens. If Null, then the refresh count (qvrcnt) is used to determine if the open was a full open (0) or a pseudo open (>0). Note: While using an SQL Plan Cache snapshot, this value can be set by the database monitor. This value might be null if the query never completed, or was running when the snapshot was created. If there is not a plan cache snapshot, the value is null.
Proc_In_Cache	QVC1F	Procedure definition was found in an internal cache. (Y/N) Only applicable for CALL statements.
Combined_Operation	QWC1A	Statement was performed with the processing for another statement. (Y/N) Only applicable for OPEN, FETCH, and CLOSE statements.

Table 130. QQQ1000 - SQL Information (continued)

View Column Name	Table Column Name	Description
Client_Applname	QVC3001	Client Special Register - application name
Client_Userid	QVC3002	Client Special Register - userid
Client_Wrkstnname	QVC3003	Client Special Register - work station name
Client_Acctng	QVC3005	Client Special Register - accounting string
Client_Programid	QVC3006	Client Special Register - program name
Interface_Information	QVC5001	Part of the CLIENT special register information. Three types of info are stored in this char500 column, separated by colons. • First part, Interface Name, varchar(127); • Second part, Interface Level, varchar(63); • Third part, Interface Type, varchar(63)
Open_Options	QVC82	Open options appear as a combination of the following characters, representing the actual capability for the cursor. The character values are left-aligned and padded on the right with blanks. Example 'RU ' indicate that the cursor is both read and update capable. • R - Read capable • W - Write capable • U - Update capable • D - Delete capable
Extended_Indicators	QWC1D	An Update or Insert statement was enabled to use extended indicators (Y/N).
DECFLOAT_Rounding_Mode	QWC1C	Rounding mode to use for DECFLOAT computations and conversions. • 'E' = ROUND_HALF_EVEN • 'C' = ROUND_CEILING • 'D' = ROUND_DOWN • 'F' = ROUND_FLOOR • 'G' = ROUND_HALF_DOWN • 'H' = ROUND_HALF_UP • 'U' = ROUND_UP
SQL_DECFLOAT_Warnings	QWC1E	DECFLOAT computations and conversions involving division by 0, overflow, underflow, an invalid operand, an inexact result, or a subnormal number results in a warning (Y/N).
Worst_Time_Micro	QVP15D	If not null, this time is the time for the slowest single run of this query. Note: When monitor files are created when using an SQL Plan Cache snapshot, this time represents the run time for the longest single run of the query. If the value is null, then the longest run information is not available. In that case, QQI6 might be the next best answer. See documentation for QQI6 for the proper use of that field
SQ_Unique_Count	QQINT05	A unique count used to uniquely identify statements which do not have an ODP but do pass in host variables. If QQUCNT is 0 and the statement passes in host variables, this value is non-zero. An example would be a CALL statement.

View Column Name	Table Column Name	Description
Concurrent_Access_Res_Used	QFC13	Specifies what method of concurrent access resolution was used. • 'N' = Concurrent access resolution is not applicable. This method applies to read queries with no commit or uncommitted read. • 'S' = SKIP LOCKED DATA clause was specified and rows with incompatible locks held by other transactions are skipped. • 'U' = USE CURRENTLY COMMITTED clause was specified and the currently committed version of data being updated or deleted is used. Data being inserted is skipped. • 'W' = Wait for commit or rollback when data is in the process of being inserted, updated, or deleted. This is the default method when the isolation level does not apply, the query is processed by CQE, or when not specified by the user.
SQL_Scalar_UDFs_Not_Inlined	QQSMINT8	Specifies the number of SQL scalar user-defined functions (UDFs) that were not inlined in an SQL query or expression.
Result_Set_Cursor	QVC3007	Result Set Cursor name. Set by Allocate Cursor, Fetch, and Close.
Implicit_XMLPARSE_Option	QFC12	CURRENT IMPLICIT XMLPARSE OPTION special register. This option is used to specify white-space handling for an implicit parse of serialized XML data. • 'S' = STRIP WHITESPACE • 'P' = PRESERVE WHITESPACE
SQL_XML_Data_CCSID	QQSMINT7	The CCSID used for XML columns, host variables, parameter markers, and expressions if not explicitly specified.
OPTIMIZER_USE	QQSMINT5	Which optimizer was used for the query. Set to null if the monitor predates this option. • 0 = Does not apply for this statement • 1 = SQE was used (SQL Query Engine) • 2 = CQE was used (Classic Query Engine) • 3 = CQE direct was used (statements like INSERT W/VALUES)
XML_Schema_In_Cache	QFC14	The XML schema binary used during XMLVALIDATE or decomposition was found in the XML cache. • 'Y' = Yes • 'N' = No
Current_User	QQC105	The value of the CURRENT USER special register. The value only appears in the QQC105 column if the SQL statement used CURRENT USER.
Row_Column_Access_Control	QFC15	Type of row or column access applied. • ' ' = Not applicable • 'C' = Column Access Control • 'R' = Row Access Control • 'B' = Both Row and Column Access Control

Table 130. QQQ1000 - SQL Information (continued)

Database monitor view 3000 - Table Scan

Displays the SQL logical view format for database monitor QQQ3000

```
Create View QQQ3000 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
```

```
        QQI9 as Thread_ID,
        QQUCNT as Unique_Count,
        QQUDEF as User_Defined,
        QQQDTN as Unique_SubSelect_Number,
        QQQDTL as SubSelect_Nested_Level,
        QQMATN as Materialized_View_Subselect_Number,
        QQMATL as Materialized_View_Nested_Level,
        QVP15E as Materialized_View_Union_Level,
        QVP15A as Decomposed_Subselect_Number,
        QVP15B as Total_Number_Decomposed_SubSelects,
        QVP15C as Decomposed_SubSelect_Reason_Code,
        QVP15D as Starting_Decomposed_SubSelect,
        QQTLN as System_Table_Schema,
        QQTFN as System_Table_Name,
        QQTMN as Member_Name,
        QQPTLN as System_Base_Table_Schema,
        QQPTFN as System_Base_Table_Name,
        QQPTMN as Base_Member_Name,
        QQTOTR as Table_Total_Rows,
        QQREST as Estimated_Rows_Selected,
        QQAJN as Estimated_Join_Rows,
        QQEPT as Estimated_Processing_Time,
        QQJNP as Join_Position,
        QQI1 as DataSpace_Number,
        QQC21 as Join_Method,
        QQC22 as Join_Type,
        QQC23 as Join_Operator,
        QQI2 as Index_Advised_Columns_Count,
        QQDSS as DataSpace_Selection,
        QQIDXA as Index_Advised,
        QQRCOD as Reason_Code,
        QQIDXD as Index_Advised_Columns,
        QVQTBL as Table_Name,
        QVQLIB as Table_Schema,
        QVPTBL as Base_Table_Name,
        QVPLIB as Base_Table_Schema,
        QVBNDY as Bound,
        QVRCNT as Unique_Refresh_Counter,
        QVJFANO as Join_Fanout,
        QVFILES as Join_Table_Count,
        QVPARPF as Parallel_Prefetch,
        QVPARPL as Parallel_PreLoad,
        QVPARD as Parallel_Degree_Requested,
        QVPARU as Parallel_Degree_Used,
        QVPARRC as Parallel_Degree_Reason_Code,
        QVCTIM as Estimated_Cumulative_Time,
        QQC11 as Skip_Sequential_Table_Scan,
        QQI3 as Table_Size,
        QVC3001 as DataSpace_Selection_Columns,
        QQC14 as Derived_Column_Selection,
        QVC3002 as Derived_Column_Selection_Columns,
        QQC18 as Read_Trigger,
        QVP157 as UDTF_Cardinality,
        QVC1281 as UDTF_Specific_Name,
        QVC1282 as UDTF_Specific_Schema,
        QVP154 as Pool_Size,
        QVP155 as Pool_Id,
        QQC13 as MQT_Replacement,
        QQC15 as InsertTable,
        QQSMINTF as Plan_Iteration_Number
        QQF1 as Average_Read_Time
 FROM   UserLib/DBMONTABLE
 WHERE  QQRID=3000)
```

Table 131. QQQ3000 - Table Scan

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name

View Column Name	Table Column Name	Description
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member name of base table
Table_Total_Rows	QQTOTR	Total rows in table
Estimated_Rows_Selected	QQREST	Estimated number of rows selected
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI1	Dataspace number

Table 131. QQQ3000 - Table Scan (continued)

View Column Name	Table Column Name	Description
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Index_Advised_Columns_Count	QQI2	Number of advised columns that use index scan-key positioning
DataSpace_Selection	QQDSS	Dataspace selection • Y - Yes • N - No
Index_Advised	QQIDXA	Index advised • Y - Yes • N - No
Reason_Code	QQRCOD	Reason code • T1 - No indexes exist. • T2 - Indexes exist, but none can be used. • T3 - Optimizer chose table scan over available indexes.
Index_Advised_Columns	QQIDXD	Columns for the index advised
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Schema of base table, long name

Table 131. QQQ3000 - Table Scan (continued)

View Column Name	Table Column Name	Description
Bound	QVBNDY	I/O or CPU bound. Possible values are: • I - I/O bound • C - CPU bound
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (Y/N)
Parallel_Degree_Requested	QVPARD	Parallel degree requested
Parallel_Degree_Used	QVPARU	Parallel degree used
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Skip_Sequential_Table_Scan	QQC11	Skip sequential table scan (Y/N)
Table_Size	QQI3	Size of table being queried
DataSpace_Selection_Columns	QVC3001	Columns used for dataspace selection
Derived_Column_Selection	QQC14	Derived column selection (Y/N)
Derived_Column_Selection_Columns	QVC3002	Columns used for derived column selection
Read_Trigger	QQC18	Read Trigger (Y/N)
UDTF_Cardinality	QVP157	User-defined table function Cardinality
UDTF_Specific_Name	QVC1281	User-defined table function specific name
UDTF_Specific_Schema	QVC1282	User-defined table function specific schema
Pool_Size	QVP154	Memory pool size
Pool_Id	QVP155	Memory pool ID
MQT_Replacement	QQC13	Materialized Query Table replaced queried table (Y/N)
Insert_Table	QQC15	This is a target table of an insert (Y/N)
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1
Average_Read_Time	QQF1	Average disk I/O time for this object

Table 131. QQQ3000 - Table Scan (continued)

Database monitor view 3001 - Index Used

Displays the SQL logical view format for database monitor QQQ3001

```
Create View QQQ3001 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQTLN as System_Table_Schema,
          QQTFN as System_Table_Name,
          QQTMN as Member_Name,
          QQPTLN as System_Base_Table_Schema,
          QQPTFN as System_Base_Table_Name,
          QQPTMN as Base_Member_Name,
          QQILNM as System_Index_Schema,
          QQIFNM as System_Index_Name,
          QQIMNM as Index_Member_Name,
          QQTOTR as Table_Total_Rows,
          QQREST as Estimated_Rows_Selected,
          QQFKEY as Index_Probe_Keys,
          QQKSEL as Index_Scan_Keys,
          QQAJN as Estimated_Join_Rows,
          QQEPT as Estimated_Processing_Time,
          QQJNP as Join_Position,
          QQI1 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QQI2 as Index_Advised_Probe_Count,
          QQKP as Index_Probe_Used,
          QQI3 as Index_Probe_Column_Count,
          QQKS as Index_Scan_Used,
          QQDSS as DataSpace_Selection,
          QQIDXA as Index_Advised,
          QQRCOD as Reason_Code,
          QQIDXD as Index_Advised_Columns,
          QQC11 as Constraint,
          QQ1000 as Constraint_Name,
          QVQTBL as Table_Name,
          QVQLIB as Table_Schema,
          QVPTBL as Base_Table_Name,
          QVPLIB as Base_Table_Schema,
          QVINAM as Index_Name,
          QVILIB as Index_Schema,
          QVBNDY as Bound,
          QVRCNT as Unique_Refresh_Counter,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_Preload,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QVCTIM as Estimated_Cumulative_Time,
          QVc14 as Index_Only_Access,
          QQc12 as Index_Fits_In_Memory,
          QQC15 as Index_Type,
          QVC12 as Index_Usage,
          QQI4 as Index_Entries,
          QQI5 as Unique_Keys,
          QQI6 as Percent_Overflow,
          QQI7 as Vector_Size,
          QQI8 as Index_Size,
```

```
          QQIA as Index_Page_Size,
          QVP154 as Pool_Size,
          QVP155 as Pool_Id,
          QVP156 as Table_Size,
          QQC16 as Skip_Sequential_Table_Scan,
          QVC13 as Tertiary_Indexes_Exist,
          QVC3001 as DataSpace_Selection_COlumns,
          QQC14 as Derived_Column_Selection,
          QVC3002 as Derived_Column_Selection_Columns,
          QVC3003 as Table_Columns_For_Index_Probe,
          QVC3004 as Table_Columns_For_Index_Scan,
          QVC3005 as Join_Selection_Columns,
          QVC3006 as Ordering_Columns,
          QVC3007 as Grouping_Columns,
          QQC18 as Read_Trigger,
          QVP157 as UDTF_Cardinality,
          QVC1281 as UDTF_Specific_Name,
          QVC1282 as UDTF_Specific_Schema,
          QQC13 as MQT_Replacement,
          QQSMINTF as Plan_Iteration_Number,
          QVC3008 as Include_Values,
          QVC15 as Sparse_Index
          QQF1 as Average_Read_Time
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3001)
```

Table 132. QQQ3001 - Index Used

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User-defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code

Table 132. QQQ3001 - Index Used (continued)

View Column Name	Table Column Name	Description
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member name of base table
System_Index_Schema	QQILNM	Schema name of index used for access
System_Index_Name	QQIFNM	Name of index used for access
Index_Member_Name	QQIMNM	Member name of index used for access
Table_Total_Rows	QQTOTR	Total rows in base table
Estimated_Rows_Selected	QQREST	Estimated number of rows selected
Index_Probe_Keys	QQFKEY	Columns selected through index scan-key positioning
Index_Scan_Keys	QQKSEL	Columns selected through index scan-key selection
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI1	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE- Less than or equal • CP - Cartesian product

View Column Name	Table Column Name	Description
Index_Advised_Probe_Count	QQI2	Number of advised key columns that use index scan-key positioning
Index_Probe_Used	QQKP	Index scan-key positioning • Y - Yes • N - No
Index_Probe_Column_Count	QQI3	Number of columns that use index scan-key positioning for the index used
Index_Scan_Used	QQKS	Index scan-key selection • Y - Yes • N - No
DataSpace_Selection	QQDSS	Dataspace selection • Y - Yes • N - No
Index_Advised	QQIDXA	Index advised • Y - Yes • N - No
Reason_Code	QQRCOD	Reason code • I1 - Row selection • I2 - Ordering/Grouping • I3 - Row selection and Ordering/Grouping • I4 - Nested loop join • I5 - Row selection using bitmap processing
Index_Advised_Columns	QQIDXD	Columns for index advised
Constraint	QQC11	Index is a constraint (Y/N)
Constraint_Name	QQ1000	Constraint name
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Schema of base table, long name
Index_Name	QVINAM	Name of index (or constraint) used, long name
Index_Schema	QVILIB	Library of index used, long name
Bound	QVBNDY	I/O or CPU bound. Possible values are: • I - I/O bound • C - CPU bound
Unique_Refresh_Counter	QVRCNT	Unique refresh counter

Table 132. QQQ3001 - Index Used (continued)

View Column Name	Table Column Name	Description
Join_Fanout	QVJFANO	Join fanout. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_Preload	QVPARPL	Parallel Preload (Y/N)
Parallel_Degree_Requested	QVPARD	Parallel degree requested
Parallel_Degree_Used	QVPARU	Parallel degree used
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Index_Only_Access	QVC14	Index only access (Y/N)
Index_Fits_In_Memory	QQC12	Index fits in memory (Y/N)
Index_Type	QQC15	Type of Index. Possible values are: • B - Binary Radix Index • C - Constraint (Binary Radix) • E - Encoded Vector Index (EVI) • X - Query created temporary index
Index_Usage	QVC12	Index Usage. Possible values are: • P - Primary Index • T - Tertiary (AND or OR) Index
Index_Entries	QQI4	Number of index entries
Unique_Keys	QQI5	Number of unique key values
Percent_Overflow	QQI6	Percent overflow
Vector_Size	QQI7	Vector size
Index_Size	QQI8	Index size
Index_Page_Size	QQIA	Index page size
Pool_Size	QVP154	Pool size
Pool_Id	QVP155	Pool ID
Table_Size	QVP156	Table size
Skip_Sequential_Table_Scan	QQC16	Skip sequential table scan (Y/N)
Tertiary_Indexes_Exist	QVC13	Tertiary indexes exist (Y/N)

Table 132. QQQ3001 - Index Used (continued)

Table 132. QQQ3001 - Index Used (continued)

View Column Name	Table Column Name	Description
DataSpace_Selection_Columns	QVC3001	Columns used for dataspace selection
Derived_Column_Selection	QQC14	Derived column selection (Y/N)
Derived_Column_Selection_Columns	QVC3002	Columns used for derived column selection
Table_Column_For_Index_Probe	QVC3003	Columns used for index scan-key positioning
Table_Column_For_Index_Scan	QVC3004	Columns used for index scan-key selection
Join_Selection_Columns	QVC3005	Columns used for Join selection
Ordering_Columns	QVC3006	Columns used for Ordering
Grouping_Columns	QVC3007	Columns used for Grouping
Read_Trigger	QQC18	Read Trigger (Y/N)
UDTF_Cardinality	QVP157	Cardinality for user-defined table function.
UDTF_Specific_Name	QVC1281	Specific name for user-defined table function.
UDTF_Specific_Schema	QVC1282	Specific schema for user-defined table function.
MQT_Replacement	QQC13	Materialized Query Table replaced queried table (Y/N)
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1
Include_Values	QVC3008	Encoded Vector indexes only. Aggregates included as part of index creation and predetermined for Grouping query request.
Sparse_Index	QVC15	Index contains sparse selection or Select/Omit selection criteria (Y/N).
Average_Read_Time	QQF1	Average disk I/O time for this object

Database monitor view 3002 - Index Created

Displays the SQL logical view format for database monitor QQQ3002.

```
Create View QQQ3002 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQTLN as System_Table_Schema,
          QQTFN as System_Table_Name,
          QQTMN as Member_Name,
          QQPTLN as System_Base_Table_Schema,
          QQPTFN as System_Base_Table_Name,
```

```
        QQPTMN as Base_Member_Name,
        QQILNM as System_Index_Schema,
        QQIFNM as System_Index_Name,
        QQIMNM as Index_Member_Name,
        QQNTNM as NLSS_Table,
        QQNLNM as NLSS_Library,
        QQSTIM as Start_Timestamp,
        QQETIM as End_Timestamp,
        QQTOTR as Table_Total_Rows,
        QQRIDX as Created_Index_Entries,
        QQREST as Estimated_Rows_Selected,
        QQFKEY as Index_Probe_Keys,
        QQKSEL as Index_Scan_Keys,
        QQAJN as Estimated_Join_Rows,
        QQEPT as Estimated_Processing_Time,
        QQJNP as Join_Position,
        QQI1 as DataSpace_Number,
        QQC21 as Join_Method,
        QQC22 as Join_Type,
        QQC23 as Join_Operator,
        QQI2 as Index_Advised_Probe_Count,
        QQKP as Index_Probe_Used,
        QQI3 as Index_Probe_Column_Count,
        QQKS as Index_Scan_Used,
        QQDSS as DataSpace_Selection,
        QQIDXA as Index_Advised,
        QQRCOD as Reason_Code,
        QQIDXD as Index_Advised_Columns,
        QQ1000 as Created_Index_Columns,
        QVQTBL as Table_Name,
        QVQLIB as Table_Schema,
        QVPTBL as Base_Table_Name,
        QVPLIB as Base_Table_Schema,
        QVINAM as Index_Name,
        QVILIB as Index_Schema,
        QVBNDY as Bound,
        QVRCNT as Unique_Refresh_Counter,
        QVJFANO as Join_Fanout,
        QVFILES as Join_Table_Count,
        QVPARPF as Parallel_Prefetch,
        QVPARPL as Parallel_Preload,
        QVPARD as Parallel_Degree_Requested,
        QVPARU as Parallel_Degree_Used,
        QVPARRC as Parallel_Degree_Reason_Code,
        QVCTIM as Estimated_Cumulative_Time,
        QQC101 as Created_Index_Name,
        QQC102 as Created_Index_Schema,
        QQI4 as Created_Index_Page_Size,
        QQI5 as Created_Index_Row_Size,
        QQC14 as Created_Index_Used_ACS_Table,
        QQC103 as Created_Index_ACS_Table,
        QQC104 as Created_Index_ACS_Library,
        QVC13 as Created_Index_Reusable,
        QVC14 as Created_Index_Sparse,
        QVC1F as Created_Index_Type,
        QVP15F as Created_Index_Unique_EVI_Count,
        QVC15 as Permanent_Index_Created,
        QVC16 as Index_From_Index,
        QVP151 as Created_Index_Parallel_Degree_Requested,
        QVP152 as Created_Index_Parallel_Degree_Used,
        QVP153 as Created_Index_Parallel_Degree_Reason_Code,
        QVC17 as Index_Only_Access,
        QVC18 as Index_Fits_In_Memory,
        QVC1B as Index_Type,
        QQI6 as Index_Entries,
        QQI7 as Unique_Keys,
        QVP158 as Percent_Overflow,
        QVP159 as Vector_Size,
        QQI8 as Index_Size,
        QVP156 as Index_Page_Size,
        QVP154 as Pool_Size,
        QVP155 as Pool_ID,
        QVP157 as Table_Size,
        QVC1C as Skip_Sequential_Table_Scan,
        QVC3001 as DataSpace_Selection_Columns,
        QVC1E as Derived_Column_Selection,
        QVC3002 as Derived_Column_Selection_Columns,
        QVC3003 as Table_Column_For_Index_Probe,
        QVC3004 as Table_Column_For_Index_Scan,
        QQC18 as Read_Trigger,
        QQC13 as MQT_Replacement,
        QQC16 as Reused_Temporary_Index,
```

```
        QQINT03 as Estimated_Storage,
        QQSMINTF as Plan_Iteration_Number,
        QQF1 as Average_Read_Time
 FROM   UserLib/DBMONTable
 WHERE  QQRID=3002)
```

Table 133. QQQ3002 - Index Created

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member name of base table
System_Index_Schema	QQILNM	Schema name of index used for access
System_Index_Name	QQIFNM	Name of index used for access

View Column Name	Table Column Name	Description
Index_Member_Name	QQIMNM	Member name of index used for access
NLSS_Table	QQNTNM	NLSS table
NLSS_Library	QQNLNM	NLSS library
Start_Timestamp	QQSTIM	Start timestamp, when available.
End_Timestamp	QQETIM	End timestamp, when available
Table_Total_Rows	QQTOTR	Total rows in table
Created_Index_Entries	QQRIDX	Number of entries in index created
Estimated_Rows_Selected	QQREST	Estimated number of rows selected
Index_Probe_Keys	QQFKEY	Keys selected thru index scan-key positioning
Index_Scan_Keys	QQKSEL	Keys selected thru index scan-key selection
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI1	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Index_Advised_Probe_Count	QQI2	Number of advised key columns that use index scan-key positioning
Index_Probe_Used	QQKP	Index scan-key positioning • Y - Yes • N - No

Table 133. QQQ3002 - Index Created (continued)

View Column Name	Table Column Name	Description
Index_Probe_Column_Count	QQI3	Number of columns that use index scan-key positioning for the index used
Index_Scan_Used	QQKS	Index scan-key selection • Y - Yes • N - No
DataSpace_Selection	QQDSS	Dataspace selection • Y - Yes • N - No
Index_Advised	QQIDXA	Index advised • Y - Yes • N - No
Reason_Code	QQRCOD	Reason code • I1 - Row selection • I2 - Ordering/Grouping • I3 - Row selection and Ordering/Grouping • I4 - Nested loop join
Index_Advised_Columns	QQIDXD	Key columns for index advised
Created_Index_Columns	QQ1000	Key columns for index created
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Schema of base table, long name
Index_Name	QVINAM	Name of index (or constraint) used, long name
Index_Schema	QVILIB	Schema of index used, long name
Bound	QVBNDY	I/O or CPU bound. Possible values are: • I - I/O bound • C - CPU bound
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.

Table 133. QQQ3002 - Index Created (continued)

Table 133. QQQ3002 - Index Created (continued)		
View Column Name	**Table Column Name**	**Description**
Join_Table_Count	QVFILES	Number of tables joined
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_Preload	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Created_Index_Name	QQC101	Name of index created - when available
Created_Index_Schema	QQC102	Schema of index created - when available
Created_Index_Page_Size	QQI4	Page size of index created
Created_Index_Row_Size	QQI5	Row size of index created
Created_Index_Used_ACS_Table	QQC14	Index Created used Alternate Collating Sequence Table (Y/N)
Created_Index_ACS_Table	QQC103	Alternate Collating Sequence table of index created.
Created_Index_ACS_Library	QQC104	Alternate Collating Sequence library of index created.
Created_Index_Reusable	QVC13	Index created is reusable (Y/N)
Created_Index_Sparse	QVC14	Index created is sparse index (Y/N)
Created_Index_Type	QVC1F	Type of index created. Possible values: • B - Binary Radix Index • E - Encoded Vector Index (EVI)
Created_Index_Unique_EVI_Count	QVP15F	Number of unique values of index created if index created is an EVI index.
Permanent_Index_Created	QVC15	Permanent index created (Y/N)
Index_From_Index	QVC16	Index from index (Y/N)
Created_Index_Parallel_Degree_Requeste d	QVP151	Parallel degree requested (index created)
Created_Index_Parallel_Degree_Used	QVP152	Parallel degree used (index created)
Created_Index_Parallel_Degree_Reason_C ode	QVP153	Reason parallel processing was limited (index created)
Index_Only_Access	QVC17	Index only access (Y/N)
Index_Fits_In_Memory	QVC18	Index fits in memory (Y/N)

Table 133. QQQ3002 - Index Created (continued)

View Column Name	Table Column Name	Description
Index_Type	QVC1B	Type of Index. Possible values are: • B - Binary Radix Index • C - Constraint (Binary Radix) • E - Encoded Vector Index (EVI) • T - Tertiary (AND/OR) Index
Index_Entries	QQI6	Number of index entries, index used
Unique_Keys	QQI7	Number of unique key values, index used
Percent_Overflow	QVP158	Percent overflow, index used
Vector_Size	QVP159	Vector size, index used
Index_Size	QQI8	Size of index used.
Index_Page_Size	QVP156	Index page size
Pool_Size	QVP154	Pool size
Pool_ID	QVP155	Pool id
Table_Size	QVP157	Table size
Skip_Sequential_Table_Scan	QVC1C	Skip sequential table scan (Y/N)
DataSpace_Selection_Columns	QVC3001	Columns used for dataspace selection
Derived_Column_Selection	QVC1E	Derived column selection (Y/N)
Derived_Column_Selection_Columns	QVC3002	Columns used for derived column selection
Table_Columns_For_Index_Probe	QVC3003	Columns used for index scan-key positioning
Table_Columns_For_Index_Scan	QVC3004	Columns used for index scan-key selection
Read_Trigger	QQC18	Read Trigger (Y/N)
MQT_Replacement	QQC13	Materialized Query Table replaced queried table (Y/N)
Reused_Temporary_Index	QQC16	Temporary index reused (Y/N)
Estimated_Storage	QQINT03	Estimated amount of temporary storage used, in megabytes, to create the temporary index.
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1
Average_Read_Time	QQF1	Average disk I/O time for this object

Database monitor view 3003 - Query Sort

Displays the SQL logical view format for database monitor QQQ3003.

```
Create View QQQ3003 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
```

```
        QQJNUM as Job_Number,
        QQI9 as Thread_ID,
        QQUCNT as Unique_Count,
        QQUDEF as User_Defined,
        QQQDTN as Unique_SubSelect_Number,
        QQQDTL as SubSelect_Nested_Level,
        QQMATN as Materialized_View_Subselect_Number,
        QQMATL as Materialized_View_Nested_Level,
        QVP15E as Materialized_View_Union_Level,
        QVP15A as Decomposed_Subselect_Number,
        QVP15B as Total_Number_Decomposed_SubSelects,
        QVP15C as Decomposed_SubSelect_Reason_Code,
        QVP15D as Starting_Decomposed_SubSelect,
        QQSTIM as Start_Timestamp,
        QQETIM as End_Timestamp,
        QQRSS as Sorted_Rows,
        QQI1 as Sort_Space_Size,
        QQI2 as Pool_Size,
        QQI3 as Pool_Id,
        QQI4 as Internal_Sort_Buffer_Length,
        QQI5 as External_Sort_Buffer_Length,
        QQRCOD as Reason_Code,
        QQI7 as Union_Reason_Subcode,
        QVBNDY as Bound,
        QVRCNT as Unique_Refresh_Counter,
        QVPARPF as Parallel_Prefetch,
        QVPARPL as Parallel_PreLoad,
        QVPARD as Parallel_Degree_Requested,
        QVPARU as Parallel_Degree_Used,
        QVPARRC as Parallel_Degree_Reason_Code,
        QQEPT as Estimated_Processing_Time,
        QVCTIM as Estimated_Cumulative_Time,
        QQAJN as Estimated_Join_Rows,
        QQJNP as Join_Position,
        QQI6 as DataSpace_Number,
        QQC21 as Join_Method,
        QQC22 as Join_Type,
        QQC23 as Join_Operator,
        QVJFANO as Join_Fanout,
        QVFILES as Join_Table_Count,
        QQINT03 as Estimated_Storage,
        QQSMINTF as Plan_Iteration_Number
        QQF1 as Average_Read_Time
 FROM   UserLib/DBMONTable
 WHERE  QQRID=3003)
```

Table 134. QQQ3003 - Query Sort

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number

Table 134. QQQ3003 - Query Sort (continued)		
View Column Name	**Table Column Name**	**Description**
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Start_Timestamp	QQSTIM	Start timestamp, when available
End_Timestamp	QQETIM	End timestamp, when available
Sorted_Rows	QQRSS	Estimated number of rows selected or sorted.
Sort_Space_Size	QQI1	Estimated size of sort space.
Pool_Size	QQI2	Pool size
Pool_Id	QQI3	Pool id
Internal_Sort_Buffer_Length	QQI4	Internal sort buffer length
External_Sort_Buffer_Length	QQI5	External sort buffer length
Reason_Code	QQRCOD	Reason code • F1 - Query contains grouping columns (GROUP BY) from more that one table, or contains grouping columns from a secondary table of a join query that cannot be reordered. • F2 - Query contains ordering columns (ORDER BY) from more that one table, or contains ordering columns from a secondary table of a join query that cannot be reordered. • F3 - The grouping and ordering columns are not compatible. • F4 - DISTINCT was specified for the query. • F5 - UNION was specified for the query. • F6 - Query had to be implemented using a sort. Key length of more than 2000 bytes or more than 120 key columns specified for ordering.

Table 134. QQQ3003 - Query Sort (continued)

View Column Name	Table Column Name	Description
Reason_Code (continued)		• F7 - Query optimizer chose to use a sort rather than an index to order the results of the query. • F8 - Perform specified row selection to minimize I/O wait time. • FC - The query contains grouping fields and there is a read trigger on at least one of the physical files in the query.
Union_Reason_Subcode	QQI7	Reason subcode for Union: • 51 - Query contains UNION and ORDER BY • 52 - Query contains UNION ALL
Bound	QVBNDY	I/O or CPU bound. Possible values are: • I - I/O bound • C - CPU bound
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join

View Column Name	Table Column Name	Description
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Estimated_Storage	QQINT03	Estimated amount of temporary storage used, in megabytes, to create the temporary index.
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1
Average_Read_Time	QQF1	Average disk I/O time for this object

Table 134. QQQ3003 - Query Sort (continued)

Database monitor view 3004 - Temp Table

Displays the SQL logical view format for database monitor QQQ3004.

```
Create View QQQ3004 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQTLN as System_Table_Schema,
          QQTFN as System_Table_Name,
          QQTMN as Member_Name,
          QQPTLN as System_Base_Table_Schema,
          QQPTFN as System_Base_Table_Name,
          QQPTMN as Base_Member_Name,
          QQSTIM as Start_Timestamp,
          QQETIM as End_Timestamp,
          QQC11 as Has_Default_Values,
```

```
            QQTMPR as Table_Rows,
            QQRCOD as Reason_Code,
            QVQTBL as Table_Name,
            QVQLIB as Table_Schema,
            QVPTBL as Base_Table_Name,
            QVPLIB as Base_Table_Schema,
            QQC101 as Temporary_Table_Name,
            QQC102 as Temporary_Table_Schema,
            QVBNDY as Bound,
            QVRCNT as Unique_Refresh_Counter,
            QVJFANO as Join_Fanout,
            QVFILES as Join_Table_Count,
            QVPARPF as Parallel_Prefetch,
            QVPARPL as Parallel_PreLoad,
            QVPARD as Parallel_Degree_Requested,
            QVPARU as Parallel_Degree_Used,
            QVPARRC as Parallel_Degree_Reason_Code,
            QQEPT as Estimated_Processing_Time,
            QVCTIM as Estimated_Cumulative_Time,
            QQAJN as Estimated_Join_Rows,
            QQJNP as Join_Position,
            QQI6 as DataSpace_Number,
            QQC21 as Join_Method,
            QQC22 as Join_Type,
            QQC23 as Join_Operator,
            QQI2 as Temporary_Table_Row_Size,
            QQI3 as Temporary_Table_Size,
            QQC12 as Temporary_Query_Result,
            QQC13 as Distributed_Temporary_Table,
            QVC3001 as Distributed_Temporary_Data_Nodes,
            QQI7 as Materialized_Subqery_QDT_Level,
            QQI8 as Materialized_Union_QDT_Level,
            QQC14 as View_Contains_Union,
            QQINT03 as Estimated_Storage,
            QQSMINTF as Plan_Iteration_Number
            QQF1 as Average_Read_Time
    FROM    UserLib/DBMONTable
    WHERE   QQRID=3004)
```

Table 135. QQQ3004 - Temp Table

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level

View Column Name	Table Column Name	Description
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member name of base table
Start_Timestamp	QQSTIM	Start timestamp, when available
End_Timestamp	QQETIM	End timestamp, when available
Has_Default_Values	QQC11	Default values may be present in temporary • Y - Yes • N - No
Table_Rows	QQTMPR	Estimated number of rows in the temporary

Table 135. QQQ3004 - Temp Table (continued)

View Column Name	Table Column Name	Description
Reason_Code	QQRCOD	Reason code. Possible values are: • F1 - Query contains grouping columns (GROUP BY) from more that one table, or contains grouping columns from a secondary table of a join query that cannot be reordered. • F2 - Query contains ordering columns (ORDER BY) from more that one table, or contains ordering columns from a secondary table of a join query that cannot be reordered. • F3 - The grouping and ordering columns are not compatible. • F4 - DISTINCT was specified for the query. • F5 - UNION was specified for the query. • F6 - Query had to be implemented using a sort. Key length of more than 2000 bytes or more than 120 key columns specified for ordering. • F7 - Query optimizer chose to use a sort rather than an index to order the results of the query. • F8 - Perform specified row selection to minimize I/O wait time. • F9 - The query optimizer chose to use a hashing algorithm rather than an index to perform the grouping. • FA - The query contains a join condition that requires a temporary table • FB - The query optimizer creates a run-time temporary file in order to implement certain correlated group by queries. • FC - The query contains grouping fields and there is a read trigger on at least one of the physical files in the query. • FD - The query optimizer creates a runtime temporary file for a static-cursor request. • H1 - Table is a join logical file and its join type does not match the join type specified in the query. • H2 - Format specified for the logical table references more than one base table. • H3 - Table is a complex SQL view requiring a temporary table to contain the results of the SQL view. • H4 - For an update-capable query, a subselect references a column in this table which matches one of the columns being updated. • H5 - For an update-capable query, a subselect references an SQL view which is based on the table being updated. • H6 - For a delete-capable query, a subselect references either the table from which rows are to be deleted, an SQL view, or an index based on the table from which rows are to be deleted

Table 135. QQQ3004 - Temp Table (continued)

Table 135. QQQ3004 - Temp Table (continued)

View Column Name	Table Column Name	Description
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Library of base table, long name
Temporary_Table_Name	QQC101	Temporary table name
Temporary_Table_Schema	QQC102	Temporary table schema
Bound	QVBNDY	I/O or CPU bound. Possible values are: • I - I/O bound • C - CPU bound
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (Y/N)
Parallel_Degree_Requested	QVPARD	Parallel degree requested
Parallel_Degree_Used	QVPARU	Parallel degree used
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join

View Column Name	Table Column Name	Description
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Temporary_Table_Row_Size	QQI2	Row size of temporary table, in bytes
Temporary_Table_Size	QQI3	Estimated size of temporary table, in bytes.
Temporary_Query_Result	QQC12	Temporary result table that contains the results of the query. (Y/N)
Distributed_Temporary_Table	QQC13	Distributed Table (Y/N)
Distributed_Temporary_Data_Nodes	QVC3001	Data nodes of temporary table
Materialized_Subqery_QDT_Level	QQI7	Materialized subquery QDT level
Materialized_Union_QDT_Level	QQI8	Materialized Union QDT level
View_Contains_Union	QQC14	Union in a view (Y/N)
Estimated_Storage	QQINT03	Estimated amount of temporary storage used, in megabytes, to create the temporary index.
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1
Average_Read_Time	QQF1	Average disk I/O time for this object

Database monitor view 3005 - Table Locked

Displays the SQL logical view format for database monitor QQQ3005.

```
Create View QQQ3005 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
```

```
            QVP15E as Materialized_View_Union_Level,
            QVP15A as Decomposed_Subselect_Number,
            QVP15B as Total_Number_Decomposed_SubSelects,
            QVP15C as Decomposed_SubSelect_Reason_Code,
            QVP15D as Starting_Decomposed_SubSelect,
            QQTLN as System_Table_Schema,
            QQTFN as System_Table_Name,
            QQTMN as Member_Name,
            QQPTLN as System_Base_Table_Schema,
            QQPTFN as System_Base_Table_Name,
            QQPTMN as Base_Member_Name,
            QQC11 as Lock_Success,
            QQC12 as Unlock_Request,
            QQRCOD as Reason_Code,
            QVQTBL as Table_Name,
            QVQLIB as Table_Schema,
            QVPTBL as Base_Table_Name,
            QVPLIB as Base_Table_Schema,
            QQJNP as Join_Position,
            QQI6 as DataSpace_Number,
            QQC21 as Join_Method,
            QQC22 as Join_Type,
            QQC23 as Join_Operator,
            QVJFANO as Join_Fanout,
            QVFILES as Join_Table_Count,
            QVRCNT as Unique_Refresh_Counter
       FROM   UserLib/DBMONTable
       WHERE  QQRID=3005)
```

Table 136. QQQ3005 - Table Locked

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects

Table 136. QQQ3005 - Table Locked (continued)

View Column Name	Table Column Name	Description
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member name of base table
Lock_Success	QQC11	Successful lock indicator (Y/N)
Unlock_Request	QQC12	Unlock request (Y/N)
Reason_Code	QQRCOD	Reason code • L1 - UNION with *ALL or *CS with Keep Locks • L2 - DISTINCT with *ALL or *CS with Keep Locks • L3 - No duplicate keys with *ALL or *CS with Keep Locks • L4 - Temporary needed with *ALL or *CS with Keep Locks • L5 - System Table with *ALL or *CS with Keep Locks • L6 - Orderby > 2000 bytes with *ALL or *CS with Keep Locks • L9 - Unknown • LA - User-defined table function with *ALL or *CS with Keep Locks
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Schema of base table, long name
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join

Table 136. QQQ3005 - Table Locked (continued)

View Column Name	Table Column Name	Description
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Unique_Refresh_Counter	QVRCNT	Unique refresh counter

Database monitor view 3006 - Access Plan Rebuilt

Displays the SQL logical view format for database monitor QQQ3006.

```
Create View QQQ3006 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQRCOD as Reason_Code,
          QQC21 as SubCode,
          QVRCNT as Unique_Refresh_Counter,
          QQTIM1 as Last_Access_Plan_Rebuild_Timestamp,
          QQC11 as Reoptimization_Done,
```

```
        QVC22 as Previous_Reason_Code,
        QVC23 as Previous_SubCode,
        QQSMINTF as Plan_Iteration_Number
FROM    UserLib/DBMONTable
WHERE   QQRID=3006)
```

Table 137. QQQ3006 - Access Plan Rebuilt

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect

Table 137. QQQ3006 - Access Plan Rebuilt (continued)

View Column Name	Table Column Name	Description
Reason_Code	QQRCOD	Reason code why access plan was rebuilt • A1 - A table or member is not the same object as the one referenced when the access plan was last built. Some reasons they might be different are: – Object was deleted and recreated. – Object was saved and restored. – Library list was changed. – Object was renamed. – Object was moved. – Object was overridden to a different object. – This is the first run of this query after the object containing the query has been restored. – Mask or permission attributes changed for the object. • A2 - Access plan was built to use a reusable Open Data Path (ODP) and the optimizer chose to use a non-reusable ODP for this call. • A3 - Access plan was built to use a non-reusable Open Data Path (ODP) and the optimizer chose to use a reusable ODP for this call. • A4 - The number of rows in the table has changed by more than 10% since the access plan was last built. • A5 - A new index exists over one of the tables in the query • A6 - An index that was used for this access plan no longer exists or is no longer valid. • A7 - IBM i Query requires the access plan to be rebuilt because of system programming changes. • A8 - The CCSID of the current job is different than the CCSID of the job that last created the access plan. • A9 - The value of one or more of the following is different for the current job than it was for the job that last created this access plan: – date format – date separator – time format – time separator.

View Column Name	Table Column Name	Description
Reason_Code (continued)	QQRCOD	• AA - The sort sequence table specified is different than the sort sequence table that was used when this access plan was created. • AB - Storage pool changed. • AC - The system feature DB2 multisystem has been installed or removed. • AD - The value of the degree query attribute has changed. • AE - A view is either being opened by a high level language or a view is being materialized. • AF - A sequence object or user-defined type or function is not the same object as the one referred to in the access plan; or, the SQL path used to generate the access plan is different than the current SQL path. • B0 - The options specified have changed as a result of the query options file. • B1 - The access plan was generated with a commitment control level that is different in the current job. • B2 - The access plan was generated with a static cursor answer set size that is different than the previous access plan. • B3 - The query was reoptimized because this is the first run of the query after a prepare. That is, it is the first run with real actual parameter marker values. • B4 - The query was reoptimized because referential or check constraints have changed. • B5 - The query was reoptimized because MQTs have changed. • B6 - The query was reoptimized because the value of a host variable changed and the access plan is no longer valid. • B7 - The query was reoptimized because AQP determined that the query should be reoptimized. • B8 - The query was reoptimized because Expression Evaluator determined that the statement should be reoptimized
SubCode	QQC21	If the access plan rebuild reason code was A7 this two-byte hex value identifies which specific reason for A7 forced a rebuild.
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Last_Access_Plan_Rebuild_Timestamp	QQTIM1	Timestamp of last access plan rebuild

Table 137. QQQ3006 - Access Plan Rebuilt (continued)

Table 137. QQQ3006 - Access Plan Rebuilt (continued)

View Column Name	Table Column Name	Description
Reoptimization_Done	QQC11	Required optimization for this plan. • Y - Yes, plan was really optimized. • N - No, the plan was not reoptimized because of the QAQQINI option for the REOPTIMIZE_ACCESS_PLAN parameter value
Previous_Reason_Code	QVC22	Previous reason code
Previous_SubCode	QVC23	Previous reason subcode
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Database monitor view 3007 - Optimizer Timed Out

Displays the SQL logical view format for database monitor QQQ3007.

```
Create View QQQ3007 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQTLN as System_Table_Schema,
          QQTFN as System_Table_Name,
          QQTMN as Member_Name,
          QQPTLN as System_Base_Table_Schema,
          QQPTFN as System_Base_Table_Name,
          QQPTMN as Base_Member_Name,
          QQ1000 as Index_Names,
          QQC11 as Optimizer_Timed_Out,
          QQC301 as Reason_Codes,
          QVQTBL as Table_Name,
          QVQLIB as Table_Schema,
          QVPTBL as Base_Table_Name,
          QVPLIB as Base_Table_Schema,
          QQJNP as Join_Position,
          QQI6 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QVRCNT as Unique_Refresh_Counter,
          QQIDXNL as Index_Names_2,
          QQSMINTF as Plan_iteration_number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3007)
```

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member name of base table

Table 138. QQQ3007 - Optimizer Timed Out

Table 138. QQQ3007 - Optimizer Timed Out (continued)		
View Column Name	**Table Column Name**	**Description**
Index_Names	QQ1000	Names of indexes not used and reason code. 1. Access path was not in a valid state. The system invalidated the access path. 2. Access path was not in a valid state. The user requested that the access path be rebuilt. 3. Access path is a temporary access path (resides in library QTEMP) and was not specified as the file to be queried. 4. The cost to use this access path, as determined by the optimizer, was higher than the cost associated with the chosen access method. 5. The keys of the access path did not match the fields specified for the ordering/grouping criteria. For distributed file queries, the access path keys must exactly match the ordering fields if the access path is to be used when ALWCPYDTA(*YES or *NO) is specified. 6. The keys of the access path did not match the fields specified for the join criteria. 7. Use of this access path will not minimize delays when reading records from the file. The user requested to minimize delays when reading records from the file. 8. The access path cannot be used for a secondary file of the join query because it contains static select/omit selection criteria. The join-type of the query does not allow the use of select/omit access paths for secondary files. 9. File contains record ID selection. The join-type of the query forces a temporary access path to be built to process the record ID selection. 10. The user specified ignore decimal data errors on the query. This disallows the use of permanent access paths.

Table 138. QQQ3007 - Optimizer Timed Out (continued)		
View Column Name	**Table Column Name**	**Description**
Index_Names (continued)	QQ1000	• 11. The access path contains static select/omit selection criteria which is not compatible with the selection in the query. • 12. The access path contains static select/omit selection criteria whose compatibility with the selection in the query cannot be determined. Either the select/omit criteria or the query selection became too complex during compatibility processing. • 13. The access path contains one or more keys which may be changed by the query during an insert or update. • 14. The access path is being deleted or is being created in an uncommitted unit of work in another process. • 15. The keys of the access path matched the fields specified for the ordering/grouping criteria. However, the sequence table associated with the access path did not match the sequence table associated with the query. • 16. The keys of the access path matched the fields specified for the join criteria. However, the sequence table associated with the access path did not match the sequence table associated with the query. • 17. The left-most key of the access path did not match any fields specified for the selection criteria. Therefore, key row positioning cannot be performed, making the cost to use this access path higher than the cost associated with the chosen access method. • 18. The left-most key of the access path matched a field specified for the selection criteria. However, the sequence table associated with the access path did not match the sequence table associated with the query. Therefore, key row positioning cannot be performed, making the cost to use this access path higher than the cost associated with the chosen access method. • 19. The access path cannot be used because the secondary file of the join query is a select/omit logical file. The join-type requires that the select/omit access path associated with the secondary file be used or, if dynamic, that an access path be created by the system.
Optimizer_Timed_Out	QQC11	Optimizer timed out (Y/N)
Reason_Codes	QQC301	List of unique reason codes used by the indexes that timed out (each index has a corresponding reason code associated with it)

View Column Name	Table Column Name	Description
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Schema of base table, long name
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Index_Names_2	QQ1000L	Index names when the list will not fit into QQ1000. Set to null otherwise
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Table 138. QQQ3007 - Optimizer Timed Out (continued)

Database monitor view 3008 - Subquery Processing

Displays the SQL logical view format for database monitor QQQ3008.

```
Create View QQQ3008 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QQI1 as Original_QDT_Count,
          QQI2 as Merged_QDT_Count,
          QQI3 as Final_QDT_Count,
          QVRCNT as Unique_Refresh_Counter,
          QQSMINTF as PlanIterNum
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3008)
```

Table 139. QQQ3008 - Subquery Processing

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects

View Column Name	Table Column Name	Description
Original_QDT_Count	QQI1	Original number of QDTs
Merged_QDT_Count	QQI2	Number of QDTs merged
Final_QDT_Count	QQI3	Final number of QDTs
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
PlanIterNum	QQSMINTF	AQP Plan iteration number, original optimization = 1

Table 139. QQQ3008 - Subquery Processing (continued)

Database monitor view 3010 - Host Variable & ODP Implementation

Displays the SQL logical view format for database monitor QQQ3010.

```
Create View QQQ3010 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQI5 as Unqiue_Refresh_Counter2,
          QQUDEF as User_Defined,
          QQC11 as ODP_Implementation,
          QQC12 as Host_Variable_Implementation,
          QQ1000 as Host_Variable_Values,
          QVRCNT as Unique_Refresh_Counter,
          QQDBCLOB1 as DBCLOB_CCSID,
          QQI7 as DBCLOB_Length,
          QQINT05 as SQ_Unique_Count,
          QVC11 as HV_Truncated
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3010)
```

Table 140. QQQ3010 - HostVar & ODP Implementation

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
Unqiue_Refresh_Counter2	QQI5	Unique refresh counter

View Column Name	Table Column Name	Description
User_Defined	QQUDEF	User defined column
ODP_Implementation	QQC11	ODP implementation • R - Reusable ODP • N - Nonreusable ODP • ' ' - Column not used
Host_Variable_Implementation	QQC12	Host variable implementation • I - Interface supplied values (ISV) • V - Host variables treated as literals (V2) • U - Table management row positioning (UP) • S - SQL Insert/Update host variable value
Host_Variable_Values	QQ1000	Host variable values
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
DBCLOB_CCSID	QQDBCLOB1	Host variables values in a DBCLOB CCSID 1200 field
DBCLOB_Length	QQI7	Length of host variables in the DBCLOB column.
SQ_Unique_Count	QQINT05	A unique count used to uniquely identify statements which do not have an ODP but do pass in host variables. If QQUCNT is 0 and the statement passes in host variables, this value will be non-zero. An example would be a CALL statement.
HV_Truncated	QVC11	Host variable has been truncated (Y/N).

Table 140. QQQ3010 - HostVar & ODP Implementation (continued)

Database monitor view 3011 - Array Host Variables

Displays the SQL logical view format for database monitor QQQ3011.

```
Create View QQQ3011 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQ19 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQC11 as ODP_Implementation,
          QQC12 as Array_Variable_Implementation,
          QQC101 as Array_Name,
          QVRCNT as Unique_Refresh_Counter,
          QQDBCLOB1 as Array_Values,
          QQINT05 as SQ_Unique_Count,
          QVC11 AS HV_Truncated,
          QVC1281 as Array_Name,
          QVC1282 as Array_Library,
          QQI1 as Max_Cardinality,
          QQI2 as Cur_Cardinality,
          QQI3 as Index_Position
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3011)
```

Table 141. QQQ3011 - Array Host Variables

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
ODP_Implementation	QQC11	ODP implementation: • R - Reusable ODP • N - Nonreusable ODP • ' ' - Column not used
Array_Variable_Implementation	QQC12	Array variable implementation: • I - Interface supplied values (ISV) • S- SQL Insert/Update array variable value
Array_Name	QQC101	Array name generated by the optimizer. Matches the array value in the QQ1000 QQHVAR field in the 3010 record.
Unique_Refresh_Counter	QVRCNT	Unique refresh counter.
Array_Values	QQDBCLOB1	Array variables values in a DBCLOB CCSID 1200 field (max 1 MB).
SQ_Unique_Count	QQINT05	A unique count used to uniquely identify statements which do not have an ODP but do pass in Arrays. If QQUCNT is 0 and the statement passes in Arrays, this value will be non-zero. An example would be a CALL statement.
HV_Truncated	QVC11	Host variable has been truncated (Y/N).
Array_Name	QVC1281	Name of Array UDT.
Array_Library	QVC1282	Library of Array UDT.
Max_Cardinality	QQI1	Maximum cardinality of Array.
Cur_Cardinality	QQI2	Current cardinality of Array.
Index_Position	QQI3	Index position in the Array designated in the QQ1000 QQHVAR field in the 3010 record.

Database monitor view 3012 - Global Variables

Displays the SQL logical view format for database monitor QQQ3012.

```
Create View QQQ3012 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQI5 as Unique_Refresh_Counter2,
          QQUDEF as User_Defined,
          QVRCNT as Unique_Refresh_Counter,
          QQDBCLOB1 as DBCLOB_Global_Variable,
          QQINT05 as SQ_Unique_Count,
          QVC11 as GV_Truncated
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3012)
```

Table 142. QQQ3012 - Global Variables

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
Unique_Refresh_Counter2	QQI5	Unique refresh counter
User_Defined	QQUDEF	User-defined column
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
DBCLOB_Global_Variable	QQDBCLOB1	Global session variable values in a DBCLOB CCSID 1200 field.
SQ_Unique_Count	QQINT05	A unique count used to uniquely identify statements which do not have an ODP but do pass in global variables. If QQUCNT is 0 and the statement passes in global variables, this value is non-zero. An example would be a CALL statement.
GV_Truncated	QVC11	Host variable has been truncated (Y/N).

Database monitor view 3014 - Generic QQ Information

Displays the SQL logical view format for database monitor QQQ3014.

```
Create View QQQ3014 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
```

```
            QQJFLD as Join_Column,
            QQRDBN as Relational_Database_Name,
            QQSYS as System_Name,
            QQJOB as Job_Name,
            QQUSER as Job_User,
            QQJNUM as Job_Number,
            QQI9 as Thread_ID,
            QQUCNT as Unique_Count,
            QQUDEF as User_Defined,
            QQQDTN as Unique_SubSelect_Number,
            QQQDTL as SubSelect_Nested_Level,
            QQMATN as Materialized_View_Subselect_Number,
            QQMATL as Materialized_View_Nested_Level,
            QVP15E as Materialized_View_Union_Level,
            QVP15A as Decomposed_Subselect_Number,
            QVP15B as Total_Number_Decomposed_SubSelects,
            QVP15C as Decomposed_SubSelect_Reason_Code,
            QVP15D as Starting_Decomposed_SubSelect,
            QQREST as Estimated_Rows_Selected,
            QQEPT as Estimated_Processing_Time,
            QQI1 as Open_Time,
            QQORDG as Has_Ordering,
            QQGRPG as Has_Grouping,
            QQJNG as Has_Join,
            QQC22 as Join_Type,
            QQUNIN as Has_Union,
            QQSUBQ as Has_Subquery,
            QWC1F as Has_Scalar_Subselect,
            QQHSTV as Has_Host_Variables,
            QQRCDS as Has_Row_Selection,
            QQC11 as Query_Governor_Enabled,
            QQC12 as Stopped_By_Query_Governor,
            QQC101 as Open_Id,
            QQC102 as Query_Options_Library,
            QQC103 as Query_Options_Table_Name,
            QQC13 as Early_Exit,
            QVRCNT as Unique_Refresh_Counter,
            QQI5 as Optimizer_Time,
            QQTIM1 as Access_Plan_Timestamp,
            QVC11 as Ordering_Implementation,
            QVC12 as Grouping_Implementation,
            QVC13 as Join_Implementation,
            QVC14 as Has_Distinct,
            QVC15 as Is_Distributed,
            QVC3001 as Distributed_Nodes,
            QVC105 as NLSS_Table,
            QVC106 as NLSS_Library,
            QVC16 as ALWCPYDATA,
            QVC21 as Access_Plan_Reason_Code,
            QVC22 as Access_Plan_Reason_SubCode,
            QVC3002 as Summary,
            QWC16 as Last_Union_Subselect,
            QVP154 as Query_PoolSize,
            QVP155 as Query_PoolID,
            QQI2 as Query_Time_Limit,
            QVC81 as Parallel_Degree,
            QQI3 as Max_Number_of_Tasks,
            QVC17 as Apply_CHGQRYA_Remote,
            QVC82 as Async_Job_Usage,
            QVC18 as Force_Join_Order_Indicator,
            QVC19 as Print_Debug_Messages,
            QVC1A as Parameter_Marker_Conversion,
            QQI4 as UDF_Time_Limit,
            QVC1283 as Optimizer_Limitations,
            QVC1E as Reoptimize_Requested,
            QVC87 as Optimize_All_Indexes,
            QQC14 as Has_Final_Decomposed_QDT,
            QQC15 as Is_Final_Decomposed_QDT,
            QQC18 as Read_Trigger,
            QQC81 as Star_Join,
            SUBSTR(QVC23,1,1) as Optimization_Goal,
            SUBSTR(QVC24,1,1) as VE_Diagram_Type,
            SUBSTR(QVC24,2,1) as Ignore_Like_Redunant_Shifts,
            QQC23 as Union_QDT,
            QQC21 as Unicode_Normalization,
            QVP153 as Pool_Fair_Share,
            QQC82 as Force_Join_Order_Requested,
            QVP152 as Force_Join_Order_Dataspace1,
            QQI6 as No_Parameter_Marker_Reason_Code,
            QVP151 as Hash_Join_Reason_Code,
            QQI7 as MQT_Refresh_Age,
            SUBSTR(QVC42,1,1) as MQT_Usage,
```

```
          QVC43 as SQE_NotUsed_Reason_Code,
          QVP156 as Estimated_IO_Count,
          QVP157 as Estimated_Processing_Cost,
          QVP158 as Estimated_CPU_Cost,
          QVP159 as Estimated_IO_Cost,
          SUBSTR(QVC44,1,1) as Has_Implicit_Numeric_Conversion,
          QVCTIM as Accumulated_Est_Process_Time,
          QQINT01 as Query_Gov_Storage_Limit,
          QQINT02 as Estimated_Storage,
          QQINT03 as Adjusted_Temp_Storage,
          QQINT04 as Original_Cost_Estimate,
          QQI8 as Parallel_Degree_Percentage,
          QFC12 as FieldProc_Encoded_Comparison,
          QFC13 as Allow_Array_Changes_INI_Opt,
          QFC11 as SQL_Concurrent_Access_Resolution,
          QQSMINTF as Plan_Iteration_Number,
          QXC11 as Warm_IO_Requested,
          QXC12 as Warm_IO_Used,
          QXC13 as Optimization_ Goal_Override,
          QXC1E as Plan_Signature_Match,
          QXC14 as Check_HostVars,
          QXC15 as FullOptimization,
          QXC16 as Pseudo_Open_Replace_Reason,
          QXC17 as WorkloadGroup
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3014)
```

Table 143. QQQ3014 - Generic QQ Information

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User-defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects

Table 143. QQQ3014 - Generic QQ Information (continued)

View Column Name	Table Column Name	Description
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Estimated_Rows_Selected	QQREST	Estimated number of rows selected
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Open_Time	QQI1	Time spent to open cursor, in milliseconds
Has_Ordering	QQORDG	Ordering (Y/N)
Has_Grouping	QQGRPG	Grouping (Y/N)
Has_Join	QQJNG	Join Query (Y/N)
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Has_Union	QQUNIN	Union Query (Y/N)
Has_Subquery	QQSUBQ	Subquery (Y/N)
Has_Scalar_Subselect	QWC1F	Scalar Subselects (Y/N)
Has_Host_Variables	QQHSTV	Host variables (Y/N)
Has_Row_Selection	QQRCDS	Row selection (Y/N)
Query_Governor_Enabled	QQC11	Query governor enabled (Y/N)
Stopped_By_Query_Governor	QQC12	Query governor stopped the query (Y/N)
Open_Id	QQC101	Query open ID
Query_Options_Library	QQC102	Query Options library name
Query_Options_Table_Name	QQC103	Query Options file name
Early_Exit	QQC13	Query early exit value
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Optimizer_Time	QQI5	Time spent in optimizer, in milliseconds
Access_Plan_Timestamp	QQTIM1	Access Plan rebuilt timestamp, last time access plan was rebuilt.
Ordering_Implementation	QVC11	Ordering implementation. Possible values are: • I - Index • S - Sort

Table 143. QQQ3014 - Generic QQ Information (continued)

View Column Name	Table Column Name	Description
Grouping_Implementation	QVC12	Grouping implementation. Possible values are: • I - Index • H - Hash grouping
Join_Implementation	QVC13	Join Implementation. Possible values are: • N - Nested Loop join • H - Hash join • C - Combination of Nested Loop and Hash
Has_Distinct	QVC14	Distinct query (Y/N)
Is_Distributed	QVC15	Distributed query (Y/N)
Distributed_Nodes	QVC3001	Distributed nodes
NLSS_Table	QVC105	Sort Sequence Table
NLSS_Library	QVC106	Sort Sequence Library
ALWCPYDATA	QVC16	ALWCPYDTA setting
Access_Plan_Reason_Code	QVC21	Reason code why access plan was rebuilt
Access_Plan_Reason_SubCode	QVC22	Subcode why access plan was rebuilt
Summary	QVC3002	Summary of query implementation. Shows dataspace number and name of index used for each table being queried.
Last_Union_Subselect	QWC16	Last part (last QDT) of Union (Y/N)
Query_PoolSize	QVP154	Pool size
Query_PoolID	QVP155	Pool id
Query_Time_Limit	QQI2	Query time limit
Parallel_Degree	QVC81	Parallel Degree • *SAME - Do not change current setting • *NONE - No parallel processing is allowed • *I/O - Any number of tasks might be used for I/O processing. SMP parallel processing is not allowed. • *OPTIMIZE - The optimizer chooses the number of tasks to use for either I/O or SMP parallel processing. • *MAX - The optimizer chooses to use either I/O or SMP parallel processing. • *SYSVAL - Use the current system value to process the query. • *ANY - Has the same meaning as *I/O. • *NBRTASKS - The number of tasks for SMP parallel processing is specified in column QVTASKN.
Max_Number_of_Tasks	QQI3	Max number of tasks

View Column Name	Table Column Name	Description
Apply_CHGQRYA_Remote	QVC17	Apply CHGQRYA remotely (Y/N)
Async_Job_Usage	QVC82	Asynchronous job usage • *SAME - Do not change current setting • *DIST - Asynchronous jobs might be used for queries with distributed tables • *LOCAL - Asynchronous jobs might be used for queries with local tables only • *ANY - Asynchronous jobs might be used for any database query • *NONE - No asynchronous jobs are allowed
Force_Join_Order_Indicator	QVC18	Force join order (Y/N)
Print_Debug_Messages	QVC19	Print debug messages (Y/N)
Parameter_Marker_Conversion	QVC1A	Parameter marker conversion (Y/N)
UDF_Time_Limit	QQI4	User Defined Function time limit
Optimizer_Limitations	QVC1283	Optimizer limitations. Possible values: • *PERCENT followed by 2 byte integer containing the percent value • *MAX_NUMBER_OF_RECORDS followed by an integer value that represents the maximum number of rows
Reoptimize_Requested		Reoptimize access plan requested. Possible values are: • O - Only reoptimize the access plan when required. Do not reoptimize for subjective reasons. • Y - Yes, force the access plan to be reoptimized. • N - No, do not reoptimize the access plan, unless optimizer determines that it is necessary. May reoptimize for subjective reasons.
Optimize_All_Indexes		Optimize all indexes requested • *SAME - Do not change current setting • *YES - Examine all indexes • *NO - Allow optimizer to time out • *TIMEOUT - Force optimizer to time out
Has_Final_Decomposed_QDT	QQC14	Final decomposed QDT built indicator (Y/N)
Is_Final_Decomposed_QDT	QQC15	The final decomposed QDT indicator (Y/N)
Read_Trigger	QQC18	One of the files contains a read trigger (Y/N)

Table 143. QQQ3014 - Generic QQ Information (continued)

Table 143. QQQ3014 - Generic QQ Information (continued)

View Column Name	Table Column Name	Description
Star_Join	QQC81	Star join optimization requested. • *NO - Star join optimization is not performed. • *COST - The optimizer determines if any EVIs can be used for star join optimization. • *FORCE - The optimizer adds any EVIs that can be used for star join optimization.
Optimization_Goal	QVC23	Byte 1 = Optimization goal. Possible values are: • F - First I/O, optimize the query to return the first screen full of rows as quickly as possible. • A - All I/O, optimize the query to return all rows as quickly as possible.
VE_Diagram_Type	QVC24	Byte 1 = Type of Visual Explain diagram. Possible values are: • D - Detail • B - Basic
Ignore_Like_Redunant_Shifts	QVC24	Byte 2 - Ignore LIKE redundant shifts. Possible values are: • O - Optimize, the query optimizer determines which redundant shifts to ignore. • A - All redundant shifts are ignored.
Union_QDT	QQC23	Byte 1 = This QDT is part of a UNION that is contained within a view (Y/N). Byte 2 = This QDT is the last subselect of the UNION that is contained within a view (Y/N).
Unicode_Normalization	QQC21	Unicode data normalization requested (Y/N)
Pool_Fair_Share	QVP153	Fair share of the pool size as determined by the optimizer
Force_Join_Order_Requested	QQC82	Force Join Order requested. Possible values are: • *NO - The optimizer was allowed to reorder join files • *YES - The optimizer was not allowed to reorder join files as part of its optimization process • *SQL - The optimizer only forced the join order for those queries that used the SQL JOIN syntax • *PRIMARY - The optimizer was only required to force the primary dial for the join.
Force_Join_Order_Dataspace1	QVP152	Primary dial to force if Force_Join_Order_Indicator is *PRIMARY.

View Column Name	Table Column Name	Description
No_Parameter_Marker_Reason_Code	QQI6	Reason code for why Parameter Marker Conversion was not performed: 1. Argument of function must be a literal 2. LOCALTIME or LOCALTIMESTAMP 3. Duration literal in arithmetic expression 4. UPDATE query with no WHERE clause 5. BLOB literal 6. Special register in UPDATE or INSERT with values 7. Result expression for CASE 8. GROUP BY expression 9. ESCAPE character 10. Double Negative value -(-1) 11. INSERT or UPDATE with a mix of literals, parameter markers, and NULLs 12. UPDATE with a mix of literals and parameter markers 13. INSERT with VALUES containing NULL value and expressions 14. UPDATE with list of expressions 99. Parameter marker conversion disabled by QAQQINI
Hash_Join_Reason_Code	QVP151	Reason code why hash join was not used.
MQT_Refresh_Age	QQI7	Value of the MATERIALIZED_QUERY_TABLE_REFRESH_AGE duration. If the QAQQINI parameter value is set to *ANY, the timestamp duration is 99999999999999.
MQT_Usage	QVC42,1,1	Byte 1 - Contains the MATERIALIZED_QUERY_TABLE_USAGE. Possible values are: • N - *NONE - no materialized query tables used in query optimization and implementation • A - *ALL - User-maintained. Refresh-deferred query tables can be used. • U - *USER - Only user-maintained materialized query tables can be used.

Table 143. QQQ3014 - Generic QQ Information (continued)

View Column Name	Table Column Name	Description
SQE_NotUsed_Reason_Code	QVC43	SQE Not Used Reason Code. Possible values: • LF - DDS logical file specified in query definition • DK - An index with derived key or select/omit was found over a queried table • NF - Too many tables in query • NS - Not an SQL query or query not run through an SQL interface • DF - Distributed table in query • RT - Read Trigger defined on queried table • PD - Program described file in query • WC - WHERE CURRENT OF a partition table • IO - Simple INSERT query • CV - Create view statement
Estimated_IO_Count	QVP156	Estimated I/O count
Estimated_Processing_Cost	QVP157	Estimated processing cost in milliseconds
Estimated_CPU_Cost	QVP158	Estimated CPU cost in milliseconds
Estimated_IO_Cost	QVP159	Estimated I/O cost in milliseconds
Has_Implicit_Numeric_Conversion	QVC44	Byte 1: Implicit numeric conversion (Y/N)
Accumulated_Est_Process_Time	QVCTIM	Accumulated estimated processing time across all subselects, in seconds.
Query_Gov_Storage_Limit	QQINT01	Specified query governor storage limit, in megabytes
Estimated_Storage	QQINT02	Original estimated temporary storage used, in megabytes.
Adjusted_Temp_Storage	QQINT03	Adjusted temporary storage used, in Adjusted megabytes. This value accumulates the actual time and storage it took to create any temporary indexes and temporary tables. Set by CQE only.
Original_Cost_Estimate	QQINT04	Original cost estimate as determined by the CQE query optimizer. Set by CQE only.
Parallel_Degree_Percentage	QQI8	Percentage specified on Parallel_Degree *OPTIMIZE and *MAX.
FieldProc_Encoded_Comparison	QFC12	FIELDPROC_ENCODED_COMPARISON option active for this query. Specifies the amount of optimization that the optimizer might use when queried columns have attached field procedures. • 'N' - NONE • 'E' - ALLOW_ EQUAL • 'R' - ALLOW_ RANGE • 'A' - ALL

Table 143. QQQ3014 - Generic QQ Information (continued)

View Column Name	Table Column Name	Description
Allow_Array_Change_INI_Opt	QFC13	ALLOW_ARRAY_VALUE_CHANGES QAQQINI option active for this query. • 'N' - Do not allow changes to values in arrays referenced in the query to be visible after the query is opened. • 'Y' - Allow changes to values in arrays to be visible to the query while the query is running.
SQL_Concurrent_Access_Resolution	QFC11	SQL_CONCURRENT_ACCESS_RESOLUTION QAQQINI option active for this query. • 'U' - USE CURRENTLY COMMITTED • 'W' - WAIT FOR OUTCOME
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number; original optimization = 1
Warm_IO_Requested	QXC11	Warm I/O value that was requested. • 'Y' - Yes, use Warm I/O • 'N' - No, do not use Warm I/O • 'D' - Default
Warm_IO_Used	QXC12	Warm I/O values used to implement the query. • 'Y' - Yes, use Warm I/O • 'N' - No, do not use Warm I/O • 'D' - Default
Optimization_Goal_Override	QXC13	Optimization Goal Override. • 'O' - Override the specified Optimize For N Rows value and use Optimize For All Rows. • 'D' - Default, use the specified Optimize For N Rows value.
Plan_Signature_Match	QXC1E	Plan signature match. • 'Y' - New plan matched old plan it replaced; same plan signature. • 'N' - New plan different from old plan it replaced; different plan signature.
Check_HostVars	QXC14	Indicates if this query is enabled for host variable selectivity checking at pseudo-open time. • ‘N’ – No pseudo-open host variable checking enabled • ‘O’ – This query is a candidate for pseudo-open host variable checking and the QAQQINI option was *OPTIMIZE • ‘Y’ - This query is a candidate for pseudo-open host variable checking and the QAQQINI option was *YES

View Column Name	Table Column Name	Description
FullOptimization	QXC15	Plan was rebuilt (Y/N)
Pseudo_Open_Replace_Reason	QXC16	Indicates if the plan was replaced due to QAQQINI PSEUDO_OPEN_CHECK_HOST_VARS option • '0' - Plan was not replaced • '1' - Plan was replaced
WorkloadGroup	QXC17	Workload Group is in effect (Y/N)

Table 143. QQQ3014 - Generic QQ Information (continued)

Database monitor view 3015 - Statistics Information

Displays the SQL logical view format for database monitor QQQ3015.

```
Create View QQQ3015 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQTLN as System_Table_Schema,
          QQTFN as System_Table_Name,
          QQTMN as Member_Name,
          QQPTLN as System_Base_Table_Schema,
          QQPTFN as System_Base_Table_Name,
          QQPTMN as Base_Member_Name,
          QVQTBL as Table_Name,
          QVQLIB as Table_Schema,
          QVPTBL as Base_Table_Name,
          QVPLIB as Base_Table_Schema,
          QQNTNM as NLSS_Table,
          QQNLNM as NLSS_Library,
          QQC11 as Statistic_Status,
          QQI2 as Statistic_Importance,
          QQ1000 as Statistic_Columns,
          QVC1000 as Statistic_ID,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3015)
```

Table 144. QQQ3015 - Statistic Information

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created

View Column Name	Table Column Name	Description
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of the base table queried
Base_Member_Name	QQPTMN	Member name of base table
Table_Name	QVQTBL	Queried table, long name
Table_Schema	QVQLIB	Schema of queried table, long name
Base_Table_Name	QVPTBL	Base table, long name
Base_Table_Schema	QVPLIB	Schema of base table, long name
NLSS_Table	QQNTNM	NLSS table
NLSS_Library	QQNLNM	NLSS library

Table 144. QQQ3015 - Statistic Information (continued)

Table 144. QQQ3015 - Statistic Information (continued)		
View Column Name	**Table Column Name**	**Description**
Statistic_Status	QQC11	Statistic Status. Possible values are: • 'N' - No statistic • 'S' - Stale statistic • ' ' - Unknown
Statistic_Importance	QQI2	Importance of this statistic
Statistic_Columns	QQ1000	Columns for the statistic advised
Statistic_ID	QVC1000	Statistic identifier
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Database monitor view 3018 - STRDBMON/ENDDBMON

Displays the SQL logical view format for database monitor QQQ3018.

```
Create View QQQ3018 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQC11 as Monitored_Job_type,
          QQC12 as Monitor_Command,
          QQC301 as Monitor_Job_Information,
          QQ1000L as STRDBMON_Command_Text,
          QQC101 as Monitor_ID,
          QQC102 as Version_Release_Mod,
          QQC103 as Group_PTF,
          QVC11 as Initial_AQP_Processing,
          QQSMINT1 as Current_Plan_Cache_Threshold,
          QQSMINT2 as Current_Plan_Cache_Subcaches,
          QQINT01  as Number_of_Currently_Active_Queries,
          QQINT02  as Current_Plan_Cache_Size,
          QQINT03  as Current_Plan_Cache_Size_Threshold,
          QQINT04  as Number_of_SMP_Threads,
          QQINT05  as Current_Number_of_MTIs,
          QQINT06  as Number_of_Pruning_Listeners,
          QQINT07  as Number_of_Plan_Cache_Awakenings,
          QQINT08  as Number_of_Plan_Cache_Naps,
          QQINT09  as Number_Pseudo_Open_queries_Hard_Closed
          QQINT0A  as Number_of_MTIs_Created,
          QQINT0B  as Number_of_MTIs_Deleted,
          QQINT0C  as Number_AQP_Wakeups,
          QQINT0D  as Number_AQP_Plans_Replaced,
          QQINT0E  as Number_of_Active_Queries,
          QVP151   as Number_of_Plans_in_Cache,
          QVP152   as Number_of_ROQs_in_Cache,
          QVP153   as Number_of_Temp_ROQs_in_Cache,
          QVP154   as Number_of_Reuseable_ROQs_in_Cache,
          QVP155   as Size_of_Temporary_Objects_stored_in_Cache,
          QVP156   as Number_of_Plans_Built_Since_Start,
          QVP157   as Number_of_Plans_Used_ROQ,
          QVP158   as Number_of_Plans_Used_nonROQ,
          QVP159   as Number_of_Plans_Used_No_ROQ,
          QVP15A   as Number_of_Plan_Cache_Probes,
          QVP15B   as Number_of_Plans_Used_from_Cache,
          QVP15C   as Number_of_Plan_Cache_No_Matches,
          QVP15D   as Number_of_Plans_Pruned,
          QVP15E   as Number_of_Plans_Removed,
          QVP15F   as Number_of_Queries_Run_Since_Start,
          QQI1   as Number_of_Query_Full_Opens_Since_Start,
```

```
         QQI2   as Number_of_Full_Opens_Which_Reused_ROQ,
         QQI3   as Number_Full_Optimizations,
         QQI4   as Number_Reopts_with_Existing_Valid_Plans
    FROM   UserLib/DBMONTable
    WHERE  QQRID=3018)
```

Table 145. QQQ3018 - STRDBMON/ENDDBMON

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Monitored_Job_type	QQC11	Type of job monitored • C - Current • J - Job name • A - All
Monitor_Command	QQC12	Command type • S - STRDBMON • E - ENDDBMON
Monitor_Job_Information	QQC301	Monitored job information • * - Current job • Job number/User/Job name • *ALL - All jobs
STRDBMON_Command_Text	QQ1000L	STRDBMON command text.
Monitor_ID	QQC101	Monitor ID
Version_Release_Mod	QQC102	Version Release and modification level
Group_PTF	QQC103	Installed Group PTF number and level. For example, 'SF99601 3' indicates that Database Group release V6R1M0, version 3 is installed.
Initial_AQP_Processing	QVC11	Initial AQP processing. Y indicates that the monitor file has already been processed to handle the AQP plan iteration number. All iterations other than the last one have been changed to a negative number. All other values indicate that the monitor file has not been processed.
Current_PC_Threshold	QQSMINT1	Database Plan Cache threshold, a value between 1-100 to represent a percent
Current_PC_Subcaches	QQSMINT2	Number of sub-caches within the database Plan Cache
Num_Active_Queries	QQINT01	Number of queries currently active
Current_PC_Size	QQINT02	Current size of database Plan Cache, in MB
Current_PC_Size_Threshold	QQINT03	Size threshold of database Plan Cache, in MB
Number_of_SMP_Threads	QQINT04	Number of SMP Threads
Current_Number_of_MTIs	QQINT05	Number of temporary indexes
Num_of_Pruning_Monitors	QQINT06	Number of event monitors used to prune the database Plan Cache

Table 145. QQQ3018 - STRDBMON/ENDDBMON (continued)

View Column Name	Table Column Name	Description
Num_of_PC_Prunnings	QQINT07	Number of times plans were pruned from the database Plan Cache
Num_of_Plan_Cache_Naps	QQINT08	Number of times the database Plan Cache became inactive
Num_POpen_Hard_Closed	QQINT09	Number of pseudo-opened queries that were hard closed
Num_of_MTIs_Created	QQINT0A	Number of temporary indexes that were created
Num_of_MTIs_Deleted	QQINT0B	Number of temporary indexes that were deleted
Num_AQP_Active	QQINT0C	Number of times AQP became active, whether a new plan was created or not
Num_AQP_Plans_Replaced	QQINT0D	Number of plans that were rebuilt due to AQP
Num_of_Plans_in_Cache	QVP151	Number of plans in the database Plan Cache
Num_of_ROQs_in_Cache	QVP152	Number of plans with a Read Only Query (ROQ) in the database Plan Cache
Num_of_Temp_ROQs_in_Cache	QVP153	Number of plans with temporary ROQs in the database Plan Cache
Num_of_Reuseable_ROQs	QVP154	Number of plans with reusable ROQs in the database Plan Cache
Size_Temp_Objects_in_Cache	QVP155	Size of temporary objects in the database Plan Cache, in MB
Num_Plans_Built_Since_Start	QVP156	Number of new plans built
Num_Plans_Used_ROQ	QVP157	Number of times a plan with a reusable ROQ was run
Num_Plans_Used_nonROQ	QVP158	Number of times a plan with a non-reusable ROQ was run
Num_Plans_Used_No_ROQ	QVP159	Number of times a plan without a ROQ was run
Num_Plan_Cache_Probes	QVP15A	Number of times the database Plan Cache was probed in hopes of finding a matching plan
Num_Plans_Used_from_Cache	QVP15B	Number of times a matching plan was found in the database Plan Cache
Num_Plan_Cache_No_Matches	QVP15C	Number of times a matching plan was not found in the database Plan Cache
Num_Plans_Pruned	QVP15D	Number of plans that were removed from the database Plan Cache due to size restrictions
Num_Plans_Removed	QVP15E	Number of obsolete plans that were removed from the database Plan Cache
Num_Run_Since_Start	QVP15F	Number of queries that were run
Num_FullOpens_Since_Start	QQI1	Number of queries that performed a full open when they were run
Num_FullOpens_Reused_ROQ	QQI2	Number of queries that performed a full open and reused an existing ROQ when run
Num_Full_Optimizations	QQI3	Number of queries that required a full optimization when run
Num_Valid_Plan_Reopts	QQI4	Number of queries that performed a full optimization even when a valid plan existed

Database monitor view 3019 - Rows retrieved

Displays the SQL logical view format for database monitor QQQ3019.

```
Create View QQQ3019 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
```

```
            QQMATL as Materialized_View_Nested_Level,
            QVP15E as Materialized_View_Union_Level,
            QVP15A as Decomposed_Subselect_Number,
            QVP15B as Total_Number_Decomposed_SubSelects,
            QVP15C as Decomposed_SubSelect_Reason_Code,
            QVP15D as Starting_Decomposed_SubSelect,
            QQI1 as CPU_Time_to_Return_All_Rows,
            QQI2 as Clock_Time_to_Return_All_Rows,
            QQI3 as Number_Synchronous_Database_Reads,
            QQI4 as Number_Synchronous_Database_Writes,
            QQI5 as Number_Asynchronous_Database_Reads,
            QQI6 as Number_Asynchronous_Database_Writes,
            QVP151 as Number_Page_Faults,
            QQI7 as Number_Rows_Returned,
            QQI8 as Number_of_Calls_for_Returned_Rows,
            QVP15F as Number_of_Times_Statement_was_Run,
            QQINT03 as Temporary_Storage,
            QQC11 as DBMON_Temp_Result_Reused,
            QQC21 as DBMON_Temp_Reused_RC,
            QQINT01 as DBMON_Temp_Reuse_Count,
            QQIA as Skip_Lock_Row_Count,
            QQINT05 as Skip_Lock_Row_Runs,
            QQINT06 as Skip_Lock_On_Runs,
            QQF1 as Adjusted_Average_Run_Time,
            QVRCNT as Unique_Refresh_Counter,
            QVP152 as Committed_Journal_Search_Requests,
            QVP153 as Committed_Journal_Search_Failures,
            QVP154 as Committed_Journal_Search_Time_Limit
    FROM    UserLib/DBMONTable
    WHERE   QQRID=3019)
```

Table 146. QQQ3019 - Rows retrieved

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects

View Column Name	Table Column Name	Description
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
CPU_Time_to_Return_All_Rows	QQI1	CPU time to return all rows, in milliseconds
Clock_Time_to_Return_All_Rows	QQI2	Clock time to return all rows, in milliseconds
Number_Synchronous_Database_Reads	QQI3	Number of synchronous database reads
Number_Synchronous_Database_Writes	QQI4	Number of synchronous database writes
Number_Asynchronous_Database_Reads	QQI5	Number of asynchronous database reads
Number_Asynchronous_Database_Writes	QQI6	Number of asynchronous database writes
Number_Page_Faults	QVP151	Number of page faults
Number_Rows_Returned	QQI7	Number of rows returned
Number_of_Calls_for_Returned_Rows	QQI8	Number of calls to retrieve rows returned
Number_of_Times_Statement_was_Run	QVP15F	Number of times this Statement was run. If Null, then the statement was run once.
Temporary_Storage	QQINT03	Amount of temporary storage used.
DBMON_Temp_Result_Reused	QQC11	Indicates if the query temporary result was reused (Y/N).
DBMON_Temp_Reused_RC	QQC21	Reason code why the query temporary result was reused.
DBMON_Temp_Reuse_Count	QQINT01	Number of times the query temporary result was reused.
Skip_Lock_Row_Count	QQIA	Number of locked rows that were skipped.
Skip_Lock_Row_Runs	QQINT05	Number of runs where some rows were skipped.
Skip_Lock_On_Runs	QQINT06	Number of runs where Skip Lock was active.
Adjusted_Average_Run_Time	QQF1	Average runtime for the query, adjusted to not include individual runs that are well outside the norm. Units in microseconds.
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Committed_Journal_Search_Requests	QVP152	Number of times the database manager searched the journal for the currently committed version of a record.
Committed_Journal_Search_Failures	QVP153	Number of times the database manager failed to find the currently committed version of a record in the journal.
Committed_Journal_Search_Time_Limit	QVP154	Maximum amount of time allowed to search the journal for the currently committed version of a record.

Table 146. QQQ3019 - Rows retrieved (continued)

Database monitor view 3020 - Index advised (SQE)

Displays the SQL logical view format for database monitor QQQ3020.

```
Create View QQQ3020 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QQTLN as System_Table_Schema,
          QQTFN as System_Table_Name,
          QQTMN as Member_Name,
          QQPTLN as System_Base_Table_Schema,
          QQPTFN as System_Base_Table_Name,
          QQPTMN as Base_Member_Name,
          QVPLIB as Base_Table_Schema,
          QVPTBL as Base_Table_Name,
          QQTOTR as Table_Total_Rows,
          QQEPT as Estimated_Processing_Time,
          QQIDXA as Index_is_Advised,
          QQIDXD as Index_Advised_Columns_Short_List,
          QQ1000L as Index_Advised_Columns_Long_List,
          QQI1 as Number_of_Advised_Columns,
          QQI2 as Number_of_Advised_Primary_Columns,
          QQRCOD as Reason_Code,
          QVRCNT as Unique_Refresh_Counter,
          QVC1F as Type_of_Index_Advised,
          QQNTNM as NLSS_Table,
          QQNLNM as NLSS_Library,
          QQSMINTF as Plan_Iteration_Number,
          QQ13 as DEPENDENT_ADVICE_ID
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3020)
```

Table 147. QQQ3020 - Index advised (SQE)

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)

Table 147. QQQ3020 - Index advised (SQE) (continued)		
View Column Name	**Table Column Name**	**Description**
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
System_Table_Schema	QQTLN	Schema of table queried
System_Table_Name	QQTFN	Name of table queried
Member_Name	QQTMN	Member name of table queried
System_Base_Table_Schema	QQPTLN	Schema name of base table
System_Base_Table_Name	QQPTFN	Name of base table for table queried
Base_Member_Name	QQPTMN	Member of base table
Base_Table_Schema	QVPLIB	Schema of base table, long name
Base_Table_Name	QVPTBL	Base table, long name
Table_Total_Rows	QQTOTR	Number of rows in the table
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Index_is_Advised	QQIDXA	Index advised (Y/N)
Index_Advised_Columns_Short_List	QQIDXD	Columns for the index advised, first 1000 bytes
Index_Advised_Columns_Long_List	QQ1000L	Column for the index advised
Number_of_Advised_Columns	QQI1	Number of indexes advised
Number_of_Advised_Primary_Columns	QQI2	Number of advised columns that use index scan-key positioning

View Column Name	Table Column Name	Description
Reason_Code	QQRCOD	Reason code • I1 - Row selection • I2 - Ordering/Grouping • I3 - Row selection and Ordering/Grouping • I5 - Row selection using bitmap processing • I6 - Source of statistics
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Type_of_Index_Advised	QVC1F	Type of index advised. Possible values are: • B - Radix index • E - Encoded vector index
NLSS_Table	QQNTNM	Sort Sequence Table
NLSS_Library	QQNLNM	Sort Sequence Library
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1
Dependent_Advice_ID	QQ13	Unique identifier used to link together OR predicate index advice

Table 147. QQQ3020 - Index advised (SQE) (continued)

Related reference

Index advisor
The query optimizer analyzes the row selection in the query and determines, based on default values, if creation of a permanent index improves performance. If the optimizer determines that a permanent index might be beneficial, it returns the key columns necessary to create the suggested index.

Database monitor view 3021 - Bitmap Created

Displays the SQL logical view format for database monitor QQQ3021.

```
Create View QQQ3021 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
```

```
          QQREST as Estimated_Rows_Selected,
          QQAJN as Estimated_Join_Rows,
          QQJNP as Join_Position,
          QQI6 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QQI2 as Bitmap_Size,
          QVP151 as Bitmap_Count,
          QVC3001 as Bitmap_IDs,
          QQINT03 as Storage_Estimate,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3021)
```

Table 148. QQQ3021 - Bitmap Created

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)

View Column Name	Table Column Name	Description
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated rows selected
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Bitmap_Size	QQI2	Bitmap size

Table 148. QQQ3021 - Bitmap Created (continued)

Table 148. QQQ3021 - Bitmap Created (continued)

View Column Name	Table Column Name	Description
Bitmap_Count	QVP151	Number of bitmaps created
Bitmap_IDs	QVC3001	Internal bitmap IDs
Storage_Estimate	QQINT03	Estimated amount of temporary storage used, in megabytes, to create the temporary index.
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1.

Database monitor view 3022 - Bitmap Merge

Displays the SQL logical view format for database monitor QQQ3022

```
Create View QQQ3022 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
          QQREST as Estimated_Rows_Selected,
          QQAJN as Estimated_Join_Rows,
          QQJNP as Join_Position,
          QQI6 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QQI2 as Bitmap_Size,
          QVC101 as Bitmap_ID,
          QVC3001 as Bitmaps_Merged,
          QQINT03 as Storage_Estimate,
            QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3022)
```

Table 149. QQQ3022 - Bitmap Merge

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification

Table 149. QQQ3022 - Bitmap Merge (continued)		
View Column Name	**Table Column Name**	**Description**
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated rows selected
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position - when available

View Column Name	Table Column Name	Description
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Bitmap_Size	QQI2	Bitmap size
Bitmap_ID	QVC101	Internal bitmap ID
Bitmaps_Merged	QVC3001	IDs of bitmaps merged together
Storage_Estimate	QQINT03	Estimated amount of temporary storage used, in megabytes, to create the final bitmap. Only set by CQE
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Table 149. QQQ3022 - Bitmap Merge (continued)

Database monitor view 3023 - Temp Hash Table Created
Displays the SQL logical view format for database monitor QQQ3023.

```
Create View QQQ3023 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
```

```
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
          QQREST as Estimated_Rows_Selected,
          QQAJN as Estimated_Join_Rows,
          QQJNP as Join_Position,
          QQI6 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QVC1F as HashTable_ReasonCode,
          QQI2 as HashTable_Entries,
          QQI3 as HashTable_Size,
          QQI4 as HashTable_Row_Size,
          QQI5 as HashTable_Key_Size,
          QQIA as HashTable_Element_Size,
          QQI7 as HashTable_PoolSize,
          QQI8 as HashTable_PoolID,
          QVC101 as HashTable_Name,
          QVC102 as HashTable_Library,
          QVC3001 as HashTable_Columns,
          QQINT03 as Storage_Estimate,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3023)
```

Table 150. QQQ3023 - Temp Hash Table Created

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column

Table 150. QQQ3023 - Temp Hash Table Created (continued)

View Column Name	Table Column Name	Description
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated rows selected
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join

Table 150. QQQ3023 - Temp Hash Table Created (continued)

View Column Name	Table Column Name	Description
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
HashTable_ReasonCode	QVC1F	Hash table reason code • J - Created for hash join • G - Created for hash grouping
HashTable_Entries	QQI2	Hash table entries
HashTable_Size	QQI3	Hash table size
HashTable_Row_Size	QQI4	Hash table row size
HashTable_Key_Size	QQI5	Hash table key size
HashTable_Element_Size	QQIA	Hash table element size
HashTable_PoolSize	QQI7	Hash table pool size
HashTable_PoolID	QQI8	Hash table pool ID
HashTable_Name	QVC101	Hash table internal name
HashTable_Library	QVC102	Hash table library
HashTable_Columns	QVC3001	Columns used to create hash table
Storage_Estimate	QQINT03	Estimated amount of temporary storage used, in megabytes, to create the hash table. Only set by CQE.
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Database monitor view 3025 - Distinct Processing

Displays the SQL logical view format for database monitor QQQ3025.

```
Create View QQQ3025 as
  (SELECT QQRID as Row_ID,
```

```
           QQTIME as Time_Created,
           QQJFLD as Join_Column,
           QQRDBN as Relational_Database_Name,
           QQSYS as System_Name,
           QQJOB as Job_Name,
           QQUSER as Job_User,
           QQJNUM as Job_Number,
           QQI9 as Thread_ID,
           QQUCNT as Unique_Count,
           QQUDEF as User_Defined,
           QQQDTN as Unique_SubSelect_Number,
           QQQDTL as SubSelect_Nested_Level,
           QQMATN as Materialized_View_Subselect_Number,
           QQMATL as Materialized_View_Nested_Level,
           QVP15E as Materialized_View_Union_Level,
           QVP15A as Decomposed_Subselect_Number,
           QVP15B as Total_Number_Decomposed_SubSelects,
           QVP15C as Decomposed_SubSelect_Reason_Code,
           QVP15D as Starting_Decomposed_SubSelect,
           QVRCNT as Unique_Refresh_Counter,
           QVPARPF as Parallel_Prefetch,
           QVPARPL as Parallel_PreLoad,
           QVPARD as Parallel_Degree_Requested,
           QVPARU as Parallel_Degree_Used,
           QVPARRC as Parallel_Degree_Reason_Code,
           QQEPT as Estimated_Processing_Time,
           QVCTIM as Estimated_Cumulative_Time,
           QQREST as Estimated_Rows_Selected,
           QQSMINTF as Plan_Iteration_Number
    FROM   UserLib/DBMONTable
    WHERE  QQRID=3025)
```

Table 151. QQQ3025 - Distinct Processing

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects

View Column Name	Table Column Name	Description
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated rows selected
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Table 151. QQQ3025 - Distinct Processing (continued)

Database monitor view 3026 - Set operation

Displays the SQL logical view format for database monitor QQQ3026.

```
Create View QQQ3026 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
          QQREST as Estimated_Rows_Selected,
          QQC11 as Union_Type,
          QVFILES as Join_Table_Count,
          QQUNIN as Has_Union,
```

```
        QWC16 as Last_Union_Subselect,
        QQC23 as Set_in_a_View,
        QQC22 as Set_Operator,
        QQSMINTF as Plan_Iteration_Number
 FROM   UserLib/DBMONTable
 WHERE  QQRID=3026)
```

Table 152. QQQ3026 - Set operatoin

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (Y/N)
Parallel_Degree_Requested	QVPARD	Parallel degree requested
Parallel_Degree_Used	QVPARU	Parallel degree used
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds

View Column Name	Table Column Name	Description
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated number of rows selected
Union_Type	QQC11	Type of union. Possible values are: • A - Union All • U - Union
Join_Table_Count	QVFILES	Number of tables queried
Has_Union	QQUNIN	Union subselect (Y/N)
Last_Union_Subselect	QWC16	This is the last subselect, or only subselect, for the query. (Y/N)
Set_in_a_View	QQC23	Set operation within a view. • Byte 1 of 2 (Y/N): This subselect is part of a query that is contained within a view and it contains a set operation (for example, Union). • Byte 2 of 2 (Y/N): This is the last subselect of the query that is contained within a view.
Set_Operator	QQC22	Type of set operation. Possible values are: • UU - Union • UA - Union All • UR - Union Recursive • EE - Except • EA - Except All • II - Intersect • IA - Intersect All
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Table 152. QQQ3026 - Set operatoin (continued)

Database monitor view 3027 - Subquery Merge
Displays the SQL logical view format for database monitor QQQ3027.

```
Create View QQQ3027 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
```

```
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
          QQREST as Estimated_Rows_Selected,
          QQAJN as Estimated_Join_Rows,
          QQJNP as Join_Position,
          QQI1 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QVP151 as Subselect_Number_of_Inner_Subquery,
          QVP152 as Subselect_Level_of_Inner_Subquery,
          QVP153 as Materialized_View_Subselect_Number_of_Inner,
          QVP154 as Materialized_View_Nested_Level_of_Inner,
          QVP155 as Materialized_View_Union_Level_of_Inner,
          QQC101 as Subquery_Operator,
          QVC21 as Subquery_Type,
          QQC11 as Has_Correlated_Columns,
          QVC3001 as Correlated_Columns,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3027)
```

Table 153. QQQ3027 - Subquery Merge

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Subselect number for outer subquery
SubSelect_Nested_Level	QQQDTL	Subselect level for outer subquery
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number for outer subquery
Materialized_View_Nested_Level	QQMATL	Materialized view subselect level for outer subquery
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects

Table 153. QQQ3027 - Subquery Merge (continued)		
View Column Name	**Table Column Name**	**Description**
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated rows selected
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position - when available
DataSpace_Number	QQI6	Dataspace number
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product

Table 153. QQQ3027 - Subquery Merge (continued)

View Column Name	Table Column Name	Description
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
Subselect_Number_of_Inner_Subquery	QVP151	Subselect number for inner subquery
Subselect_Level_of_Inner_Subquery	QVP152	Subselect level for inner subquery
Materialized_View_Subselect_Number_of_Inner	QVP153	Materialized view subselect number for inner subquery
Materialized_View_Nested_Level_of_Inner	QVP154	Materialized view subselect level for inner subquery
Materialized_View_Union_Level_of_Inner	QVP155	Materialized view union level for inner subquery
Subquery_Operator	QQC101	Subquery operator. Possible values are: • EQ - Equal • NE - Not Equal • LT - Less Than or Equal • LT - Less Than • GE - Greater Than or Equal • GT - Greater Than • IN • LIKE • EXISTS • NOT - Can precede IN, LIKE or EXISTS
Subquery_Type	QVC21	Subquery type. Possible values are: • SQ - Subquery • SS - Scalar subselect • SU - Set Update
Has_Correlated_Columns	QQC11	Correlated columns exist (Y/N)
Correlated_Columns	QVC3001	List of correlated columns with corresponding QDT number
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Database monitor view 3028 - Grouping

Displays the SQL logical view format for database monitor QQQ3028.

```
Create View QQQ3028 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
          QQREST as Estimated_Rows_Selected,
          QQAJN as Estimated_Join_Rows,
          QQJNP as Join_Position,
          QQI1 as DataSpace_Number,
          QQC21 as Join_Method,
          QQC22 as Join_Type,
          QQC23 as Join_Operator,
          QVJFANO as Join_Fanout,
          QVFILES as Join_Table_Count,
          QQC11 as GroupBy_Implementation,
          QQC101 as GroupBy_Index_Name,
          QQC102 as GroupBy_Index_Library,
          QVINAM as GroupBy_Index_Long_Name,
          QVILIB as GroupBy_Index_Long_Library,
          QQC12 as Has_Having_Selection,
          QQC13 as Having_to_Where_Selection_Conversion,
          QQI2 as Estimated_Number_of_Groups,
          QQI3 as Average_Number_Rows_per_Group,
          QVC3001 as GroupBy_Columns,
          QVC3002 as MIN_Columns,
          QVC3003 as MAX_Columns,
          QVC3004 as SUM_Columns,
          QVC3005 as COUNT_Columns,
          QVC3006 as AVG_Columns,
          QVC3007 as STDDEV_Columns,
          QVC3008 as VAR_Columns,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3028)
```

Table 154. QQQ3028 - Grouping

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name

View Column Name	Table Column Name	Description
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job user
Job_Number	QQJNUM	Job number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (index used)
Parallel_Degree_Requested	QVPARD	Parallel degree requested (index used)
Parallel_Degree_Used	QVPARU	Parallel degree used (index used)
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited (index used)
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated rows selected
Estimated_Join_Rows	QQAJN	Estimated number of joined rows
Join_Position	QQJNP	Join position
DataSpace_Number	QQI1	Dataspace number

Table 154. QQQ3028 - Grouping (continued)

Table 154. QQQ3028 - Grouping (continued)		
View Column Name	**Table Column Name**	**Description**
Join_Method	QQC21	Join method - when available • NL - Nested loop • MF - Nested loop with selection • HJ - Hash join
Join_Type	QQC22	Join type - when available • IN - Inner join • PO - Left partial outer join • EX - Exception join
Join_Operator	QQC23	Join operator - when available • EQ - Equal • NE - Not equal • GT - Greater than • GE - Greater than or equal • LT - Less than • LE - Less than or equal • CP - Cartesian product
Join_Fanout	QVJFANO	Join fan out. Possible values are: • N - Normal join situation where fanout is allowed and each matching row of the join fanout is returned. • D - Distinct fanout. Join fanout is allowed however none of the join fanout rows are returned. • U - Unique fanout. Join fanout is not allowed. Error situation if join fanout occurs.
Join_Table_Count	QVFILES	Number of tables joined
GroupBy_Implementation	QQC11	Group by implementation • ' ' - No grouping • I - Index • H - Hash
GroupBy_Index_Name	QQC101	Index, or constraint, used for grouping
GroupBy_Index_Library	QQC102	Library of index used for grouping
GroupBy_Index_Long_Name	QVINAM	Long name of index, or constraint, used for grouping
GroupBy_Index_Long_Library	QVILIB	Long name of index, or constraint, library used for grouping
Has_Having_Selection	QQC12	Having selection exists (Y/N)
Having_to_Where_Selection_Conversion	QQC13	Having to Where conversion (Y/N)

Table 154. QQQ3028 - Grouping (continued)		
View Column Name	**Table Column Name**	**Description**
Estimated_Number_of_Groups	QQI2	Estimated number of groups
Average_Number_Rows_per_Group	QQI3	Average number of rows in each group
GroupBy_Columns	QVC3001	Grouping columns
MIN_Columns	QVC3002	MIN columns
MAX_Columns	QVC3003	MAX columns
SUM_Columns	QVC3004	SUM columns
COUNT_Columns	QVC3005	COUNT columns
AVG_Columns	QVC3006	AVG columns
STDDEV_Columns	QVC3007	STDDEV columns
VAR_Columns	QVC3008	VAR columns
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Database monitor view 3030 - Materialized query tables

Displays the SQL logical view format for database monitor QQQ3030.

```
Create View QQQ3030 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QQ1000 as Materialized_Query_Tables,
          QQC301 as MQT_Reason_Codes,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3030)
```

Table 155. QQQ3030 - Materialized query tables		
View Column Name	**Table Column Name**	**Description**
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)

View Column Name	Table Column Name	Description
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job User
Job_Number	QQJNUM	Job Number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Materialized_Query_Tables	QQ1000	Materialized query tables examined and reason why used or not used: • 0 - The materialized query table was used • 1 - The cost to use the materialized query table, as determined by the optimizer, was higher than the cost associated with the chosen implementation. • 2 - The join specified in the materialized query was not compatible with the query. • 3 - The materialized query table had predicates that were not matched in the query. • 4 - The grouping specified in the materialized query table is not compatible with the grouping specified in the query.

Table 155. QQQ3030 - Materialized query tables (continued)

Table 155. QQQ3030 - Materialized query tables (continued)		
View Column Name	**Table Column Name**	**Description**
Materialized_Query_Tables (continued)		• 5 - The query specified columns that were not in the select-list of the materialized query table. • 6 - The materialized query table query contains functionality that is not supported by the query optimizer. • 7 - The materialized query table specified the DISABLE QUERY OPTIMIZATION clause. • 8 - The ordering specified in the materialized query table is not compatible with the ordering specified in the query. • 9 - The query contains functionality that is not supported by the materialized query table matching algorithm. • 10 - Materialized query tables may not be used for this query. • 11 - The refresh age of this materialized query table exceeds the duration specified by the MATERIALIZED_QUERY_TABLE_REFRESH_AGE QAQQINI option. • 12 - The commit level of the materialized query table is lower than the commit level specified for the query. • 13 - The distinct specified in the materialized query table is not compatible with the distinct specified in the query. • 14 - The FETCH FOR FIRST n ROWS clause of the materialized query table is not compatible with the query. • 15 - The QAQQINI options used to create the materialized query table are not compatible with the QAQQINI options used to run this query. • 16 - The materialized query table is not usable. • 17 - The union specified in the materialized query table is not compatible with the query. • 18 - The constants specified in the materialized query table are not compatible with host variable values specified in the query. • 19 - The Materialized query table is in check pending status. • 20 - The UDTF specified in the materialized query table was not compatible with the query. • 21 - The VALUES clause specified in the materialized query table was not compatible with the query. • 22 - The UNNEST clause specified in the materialized query table was not compatible with the query.

View Column Name	Table Column Name	Description
MQT_Reason_Codes	QQC301	List of unique reason codes used by the materialized query tables (each materialized query table has a corresponding reason code associated with it)
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Table 155. QQQ3030 - Materialized query tables (continued)

Database monitor view 3031 - Recursive common table expressions
Displays the SQL logical view format for database monitor QQQ3031.

```
Create View QQQ3031 as
  (SELECT QQRID as Row_ID,
          QQTIME as Time_Created,
          QQJFLD as Join_Column,
          QQRDBN as Relational_Database_Name,
          QQSYS as System_Name,
          QQJOB as Job_Name,
          QQUSER as Job_User,
          QQJNUM as Job_Number,
          QQI9 as Thread_ID,
          QQUCNT as Unique_Count,
          QQUDEF as User_Defined,
          QQQDTN as Unique_SubSelect_Number,
          QQQDTL as SubSelect_Nested_Level,
          QQMATN as Materialized_View_Subselect_Number,
          QQMATL as Materialized_View_Nested_Level,
          QVP15E as Materialized_View_Union_Level,
          QVP15A as Decomposed_Subselect_Number,
          QVP15B as Total_Number_Decomposed_SubSelects,
          QVP15C as Decomposed_SubSelect_Reason_Code,
          QVP15D as Starting_Decomposed_SubSelect,
          QVRCNT as Unique_Refresh_Counter,
          QVPARPF as Parallel_Prefetch,
          QVPARPL as Parallel_PreLoad,
          QVPARD as Parallel_Degree_Requested,
          QVPARU as Parallel_Degree_Used,
          QVPARRC as Parallel_Degree_Reason_Code,
          QQEPT as Estimated_Processing_Time,
          QVCTIM as Estimated_Cumulative_Time,
          QQREST as Estimated_Rows_Selected,
          QQC11 as Recursive_Query_Cycle_Check,
          QQC15 as Recursive_Query_Search_Option,
          QQI2 as Number_of_Recursive_Values,
          QQSMINTF as Plan_Iteration_Number
   FROM   UserLib/DBMONTable
   WHERE  QQRID=3031)
```

Table 156. QQQ3031 - Recursive common table expressions

View Column Name	Table Column Name	Description
Row_ID	QQRID	Row identification
Time_Created	QQTIME	Time row was created
Join_Column	QQJFLD	Join column (unique per job)
Relational_Database_Name	QQRDBN	Relational database name
System_Name	QQSYS	System name
Job_Name	QQJOB	Job name
Job_User	QQUSER	Job User

Table 156. QQQ3031 - Recursive common table expressions (continued)

View Column Name	Table Column Name	Description
Job_Number	QQJNUM	Job Number
Thread_ID	QQI9	Thread identifier
Unique_Count	QQUCNT	Unique count (unique per query)
User_Defined	QQUDEF	User defined column
Unique_SubSelect_Number	QQQDTN	Unique subselect number
SubSelect_Nested_Level	QQQDTL	Subselect nested level
Materialized_View_Subselect_Number	QQMATN	Materialized view subselect number
Materialized_View_Nested_Level	QQMATL	Materialized view nested level
Materialized_View_Union_Level	QVP15E	Materialized view union level
Decomposed_Subselect_Number	QVP15A	Decomposed query subselect number, unique across all decomposed subselects
Total_Number_Decomposed_SubSelects	QVP15B	Total number of decomposed subselects
Decomposed_SubSelect_Reason_Code	QVP15C	Decomposed query subselect reason code
Starting_Decomposed_SubSelect	QVP15D	Decomposed query subselect number for the first decomposed subselect
Unique_Refresh_Counter	QVRCNT	Unique refresh counter
Parallel_Prefetch	QVPARPF	Parallel Prefetch (Y/N)
Parallel_PreLoad	QVPARPL	Parallel Preload (Y/N)
Parallel_Degree_Requested	QVPARD	Parallel degree requested
Parallel_Degree_Used	QVPARU	Parallel degree used
Parallel_Degree_Reason_Code	QVPARRC	Reason parallel processing was limited
Estimated_Processing_Time	QQEPT	Estimated processing time, in seconds
Estimated_Cumulative_Time	QVCTIM	Estimated cumulative time, in seconds
Estimated_Rows_Selected	QQREST	Estimated number of rows selected
Recursive_Query_Cycle_Check	QQC11	CYCLE option: • Y - checking for cyclic data • N - not checking for cyclic data
Recursive_Query_Search_Option	QQC15	SEARCH option: • N - None specified • D - Depth first • B - Breadth first

Table 156. QQQ3031 - Recursive common table expressions (continued)

View Column Name	Table Column Name	Description
Number_of_Recursive_Values	QQI2	Number of values put on queue to implement recursion. Includes values necessary for CYCLE and SEARCH options.
Plan_Iteration_Number	QQSMINTF	AQP Plan iteration number, original optimization = 1

Query optimizer messages reference

See the following for query optimizer message reference:

Query optimization performance information messages

You can evaluate the structure and performance of the SQL statements in a program using informational messages. These messages are put in the job log by the database manager.

The messages are issued for an SQL program or interactive SQL when running in the debug mode. The database manager could send any of the following messages when appropriate. The ampersand variables (&1, &X) are replacement variables that contain either an object name or other substitution value when you see the message in the job log. These messages provide feedback on how a query was run. In some cases, the messages indicate improvements you can make to help the query run faster.

The messages contain message help that provides information about the cause for the message, object name references, and possible user responses.

The time at which the message is sent does not necessarily indicate when the associated function was performed. Some messages are sent altogether at the start of a query run.

CPI4321 - Access path built for &18 &19	
Message Text:	Access path built for &18 &19.

CPI4321 - Access path built for &18 &19	
Cause Text:	A temporary access path was built to access records from member &6 of &18 &19 in library &5 for reason code &10. This process took &11 minutes and &12 seconds. The access path built contains &15 entries. The access path was built using &16 parallel tasks. A zero for the number of parallel tasks indicates that parallelism was not used. The reason codes and their meanings follow: 1 - Perform specified ordering/grouping criteria. 2 - Perform specified join criteria. 3 - Perform specified record selection to minimize I/O wait time. The access path was built using the following key fields. The key fields and their corresponding sequence (ASCEND or DESCEND) will be shown: &17. A key field of *MAP indicates the key field is an expression (derived field). The access path was built using sequence table &13 in library &14. A sequence table of *N indicates the access path was built without a sequence table. A sequence table of *I indicates the table was an internally derived table that is not available to the user. If &18 &19 in library &5 is a logical file then the access path is built over member &9 of physical file &7 in library &8. A file name starting with *QUERY or *N indicates the access path was built over a temporary file.
Recovery Text:	If this query is run frequently, you may want to create an access path (index) similar to this definition for performance reasons. Create the access path using sequence table &13 in library &14, unless the sequence table is *N. If an access path is created, it is possible the query optimizer may still choose to create a temporary access path to process the query. If *MAP is returned for one of the key fields or *I is returned for the sequence table, then a permanent access path cannot be created. A permanent access path cannot be built with these specifications.

CPI4322 - Access path built from keyed file &1	
Message Text:	Access path built from keyed file &1.

CPI4322 - Access path built from keyed file &1	
Cause Text:	A temporary access path was built using the access path from member &3 of keyed file &1 in library &2 to access records from member &6 of file &4 in library &5 for reason code &10. This process took &11 minutes and &12 seconds. The access path built contains &15 entries. The reason codes and their meanings follow: 1 - Perform specified ordering/grouping criteria. 2 - Perform specified join criteria. 3 - Perform specified record selection to minimize I/O wait time. The access path was built using the following key fields. The key fields and their corresponding sequence (ASCEND or DESCEND) will be shown: &17. A key field of *MAP indicates the key field is an expression (derived field). The temporary access path was built using sequence table &13 in library &14. A sequence table of *N indicates the access path was built without a sequence table. A sequence table of *I indicates the table was an internally derived table that is not available to the user. If file &4 in library &5 is a logical file then the temporary access path is built over member &9 of physical file &7 in library &8. Creating an access path from a keyed file generally results in improved performance.
Recovery Text:	If this query is run frequently, you may want to create an access path (index) similar to this definition for performance reasons. Create the access path using sequence table &13 in library &14, unless the sequence table is *N. If an access path is created, it is possible the query optimizer may still choose to create a temporary access path to process the query. If *MAP is returned for one of the key fields or *I is returned for the sequence table, then a permanent access path cannot be created. A permanent access path cannot be built with these specifications. A temporary access path can only be created using index only access if all of the fields that were used by this temporary access path are also key fields for the access path from the keyed file.

CPI4323 - The query access plan has been rebuilt	
Message Text:	The query access plan has been rebuilt.

CPI4323 - The query access plan has been rebuilt	
Cause Text:	The access plan was rebuilt for reason code &13. The reason codes and their meanings follow: 0 - A new access plan was created. 1 - A file or member is not the same object as the one referred to in the access plan. Some reasons include the object being recreated, restored, or overridden to a new object. 2 - Access plan was using a reusable Open Data Path (ODP), and the optimizer chose to use a non-reusable ODP. 3 - Access plan was using a non-reusable Open Data Path (ODP) and the optimizer chose to use a reusable ODP. 4 - The number of records in member &3 of file &1 in library &2 has changed by more than 10%. 5 - A new access path exists over member &6 of file &4 in library &5. 6 - An access path over member &9 of file &7 in library &8 that was used for this access plan no longer exists or is no longer valid. 7 - The query access plan had to be rebuilt because of system programming changes. 8 - The CCSID (Coded Character Set Identifier) of the current job is different than the CCSID used in the access plan. 9 - The value of one of the following is different in the current job: date format, date separator, time format, or time separator. 10 - The sort sequence table specified has changed. 11 - The number of active processors or the size or paging option of the storage pool has changed. 12 - The system feature DB2 Symmetric Multiprocessing has either been installed or removed. 13 - The value of the degree query attribute has changed either by the **CHGSYSVAL** or **CHGQRYA** CL commands or with the query options file &15 in library &16. 14 - A view is either being opened by a high level language open, or is being materialized. 15 - A sequence object or user-defined type or function is not the same object as the one referred to in the access plan; or, the SQL path used to generate the access plan is different than the current SQL path. 16 - Query attributes have been specified from the query options file &15 in library &16. 17 - The access plan was generated with a commitment control level that is different in the current job. 18 - The access plan was generated with a different static cursor answer set size. 19 - This is the first run of the query since a prepare or compile. 20 and greater -- View the second level message text of the next message issued (CPI4351) for an explanation of these reason codes. If the reason code is 4, 5, 6, 20, or 21 and the file specified in the reason code explanation is a logical file, then member &12 of physical file &10 in library &11 is the file with the specified change.
Recovery Text:	Excessive rebuilds should be avoided and may indicate an application design problem.

CPI4324 - Temporary file built for file &1	
Message Text:	Temporary file built for file &1.

CPI4324 - Temporary file built for file &1	
Cause Text:	A temporary file was built for member &3 of file &1 in library &2 for reason code &4. This process took &5 minutes and &6 seconds. The temporary file was required in order for the query to be processed. The reason codes and their meanings follow: 1 - The file is a join logical file and its join-type (JDFTVAL) does not match the join-type specified in the query. 2 - The format specified for the logical file references more than one physical file. 3 - The file is a complex SQL view, or nested table expression, or common table expression, or is a data change table reference that requires a temporary file. 4 - For an update-capable query, a subselect references a field in this file which matches one of the fields being updated. 5 - For an update-capable query, a subselect references SQL view &1, which is based on the file being updated. 6 - For a delete-capable query, a subselect references either the file from which records are to be deleted or an SQL view or logical file based on the file from which records are to be deleted. 7 - The file is user-defined table function &8 in &2, and all the records were retrieved from the function. The processing time is not returned for this reason code. 8 - The file is a partition file requiring a temporary file for processing the grouping or join.
Recovery Text:	You may want to change the query to refer to a file that does not require a temporary file to be built.

CPI4325 - Temporary result file built for query	
Message Text:	Temporary result file built for query.

CPI4325 - Temporary result file built for query	
Cause Text:	A temporary result file was created to contain the results of the query for reason code &4. This process took &5 minutes and &6 seconds. The temporary file created contains &7 records. The reason codes and their meanings follow: 1 - The query contains grouping fields (GROUP BY) from more than one file, or contains grouping fields from a secondary file of a join query that cannot be reordered. 2 - The query contains ordering fields (ORDER BY) from more than one file, or contains ordering fields from a secondary file of a join query that cannot be reordered. 3 - The grouping and ordering fields are not compatible. 4 - DISTINCT was specified for the query. 5 - Set operator (UNION, EXCEPT, or INTERSECT) was specified for the query. 6 - The query had to be implemented using a sort. More than 120 key fields specified for ordering. 7 - The query optimizer chose to use a sort rather than an access path to order the results of the query. 8 - Perform specified record selection to minimize I/O wait time. 9 - The query optimizer chose to use a hashing algorithm rather than an access path to perform the grouping for the query. 10 - The query contains a join condition that requires a temporary file. 11 - The query optimizer creates a run-time temporary file in order to implement certain correlated group by queries. 12 - The query contains grouping fields (GROUP BY, MIN/MAX, COUNT, etc.) and there is a read trigger on one or more of the underlying physical files in the query. 13 - The query involves a static cursor or the SQL FETCH FIRST clause.
Recovery Text:	For more information on why a temporary result was used, refer to "Data access methods" on page 11.

CPI4325 - Temporary result file built for query	
Message Text:	&12 &13 processed in join position &10.

CPI4325 - Temporary result file built for query	
Cause Text:	Access path for member &5 of file &3 in library &4 was used to access records in member &2 of file &13 in library &1 for reason code &9. The reason codes and their meanings follow: 1 - Perform specified record selection. 2 - Perform specified ordering/grouping criteria. 3 - Record selection and ordering/grouping criteria. 4 - Perform specified join criteria. If file &13 in library &1 is a logical file then member &8 of physical file &6 in library &7 is the actual file in join position &10. A file name starting with *TEMPX for the access path indicates it is a temporary access path built over file &6. A file name starting with *N or *QUERY for the file indicates it is a temporary file. Index only access was used for this file within the query: &11. A value of *YES for index only access processing indicates that all of the fields used from this file for this query can be found within the access path of file &3. A value of *NO indicates that index only access could not be performed for this access path. Index only access is generally a performance advantage since all of the data can be extracted from the access path and the data space does not have to be paged into active memory.
Recovery Text:	Generally, to force a file to be processed in join position 1, specify an order by field from that file only. If ordering is desired, specifying ORDER BY fields over more than one file forces the creation of a temporary file and allows the optimizer to optimize the join order of all the files. No file is forced to be first. An access path can only be considered for index only access if all of the fields used within the query for this file are also key fields for that access path. Refer to the “Data access methods” on page 11 for additional tips on optimizing a query's join order and index only access. In some cases, creating a temporary result table provides the fastest way to run a query. Other queries that have many rows to be copied into the temporary result table can take a significant amount of time. However, if the query is taking more time and resources than can be allowed, consider changing the query so that a temporary result table is not required.

CPI4326 - &12 &13 processed in join position &10	
Message Text:	&12 &13 processed in join position &10.

CPI4326 - &12 &13 processed in join position &10	
Cause Text:	Access path for member &5 of file &3 in library &4 was used to access records in member &2 of file &13 in library &1 for reason code &9. The reason codes and their meanings follow: 1 - Perform specified record selection. 2 - Perform specified ordering/grouping criteria. 3 - Record selection and ordering/grouping criteria. 4 - Perform specified join criteria. If file &13 in library &1 is a logical file then member &8 of physical file &6 in library &7 is the actual file in join position &10. A file name starting with *TEMPX for the access path indicates it is a temporary access path built over file &6. A file name starting with *N or *QUERY for the file indicates it is a temporary file. Index only access was used for this file within the query: &11. A value of *YES for index only access processing indicates that all of the fields used from this file for this query can be found within the access path of file &3. A value of *NO indicates that index only access could not be performed for this access path. Index only access is generally a performance advantage since all of the data can be extracted from the access path and the data space does not have to be paged into active memory.
Recovery Text:	Generally, to force a file to be processed in join position 1, specify an order by field from that file only. If ordering is desired, specifying ORDER BY fields over more than one file forces the creation of a temporary file and allows the optimizer to optimize the join order of all the files. No file is forced to be first. An access path can only be considered for index only access if all of the fields used within the query for this file are also key fields for that access path. Refer to the “Data access methods” on page 11 for additional tips on optimizing a query's join order and index only access. In some cases, creating a temporary result table provides the fastest way to run a query. Other queries that have many rows to be copied into the temporary result table can take a significant amount of time. However, if the query is taking more time and resources than can be allowed, consider changing the query so that a temporary result table is not required.

This message provides the join position of the specified table when an index is used to access the table data. **Join position** pertains to the order in which the tables are joined.

CPI4327 - File &12 &13 processed in join position &10	
Message Text:	&12 &13 processed in join position &10.
Cause Text:	Arrival sequence access was used to select records from member &2 of file &13 in library &1. If file &13 in library &1 is a logical file then member &8 of physical file &6 in library &7 is the actual file in join position &10. A file name that starts with *QUERY for the file indicates it is a temporary file.

CPI4327 - File &12 &13 processed in join position &10	
Recovery Text:	Generally, to force a file to be processed in join position 1, specify an order by field from that file only. Refer to the “Data access methods” on page 11 for additional tips on optimizing a query's join order.

CPI4328 - Access path of file &3 was used by query	
Message Text:	Access path of file &3 was used by query.
Cause Text:	Access path for member &5 of file &3 in library &4 was used to access records from member &2 of &12 &13 in library &1 for reason code &9. The reason codes and their meanings follow: 1 - Record selection. 2 - Ordering/grouping criteria. 3 - Record selection and ordering/grouping criteria. If file &13 in library &1 is a logical file then member &8 of physical file &6 in library &7 is the actual file being accessed. Index only access was used for this query: &11. A value of *YES for index only access processing indicates that all of the fields used for this query can be found within the access path of file &3. A value of *NO indicates that index only access could not be performed for this access path. Index only access is generally a performance advantage since all of the data can be extracted from the access path and the data space does not have to be paged into active memory.
Recovery Text:	An access path can only be considered for index only access if all of the fields used within the query for this file are also key fields for that access path. Refer to the “Data access methods” on page 11. for additional tips on index only access.

CPI4329 - Arrival sequence access was used for &12 &13	
Message Text:	Arrival sequence access was used for &12 &13.
Cause Text:	Arrival sequence access was used to select records from member &2 of file &13 in library &1. If file &13 in library &1 is a logical file then member &8 of physical file &6 in library &7 is the actual file from which records are being selected. A file name starting with *N or *QUERY for the file indicates it is a temporary file.
Recovery Text:	The use of an access path may improve the performance of the query if record selection is specified. If an access path does not exist, you may want to create one whose left-most key fields match fields in the record selection. Matching more key fields in the access path with fields in the record selection will result in improved performance. Generally, to force the use of an existing access path, specify order by fields that match the left-most key fields of that access path. For more information refer to “Data access methods” on page 11.

CPI432A - Query optimizer timed out for file &1	
Message Text:	Query optimizer timed out for file &1.
Cause Text:	The query optimizer timed out before it could consider all access paths built over member &3 of file &1 in library &2. The list below shows the access paths considered before the optimizer timed out. If file &1 in library &2 is a logical file then the access paths specified are actually built over member &9 of physical file &7 in library &8. Following each access path name in the list is a reason code which explains how the optimizer considered the access path. &11. The reason codes and their meanings follow: 0 - The access path was used to implement the query. 1 - Access path was not in a valid state. The system invalidated the access path. 2 - Access path was not in a valid state. The user requested that the access path be rebuilt. 3 - Access path is a temporary access path (resides in library QTEMP) and was not specified as the file to be queried. 4 - The cost to use this access path, as determined by the optimizer, was higher than the cost associated with the chosen access method. 5 - The keys of the access path did not match the fields specified for the ordering/grouping criteria. 6 - The keys of the access path did not match the fields specified for the join criteria. 7 - Use of this access path would not minimize delays when reading records from the file as the user requested. 8 - The access path cannot be used for a secondary file of the join query because it contains static select/omit selection criteria. The join-type of the query does not allow the use of select/omit access paths for secondary files. 9 - File &1 contains record ID selection. The join-type of the query forces a temporary access path to be built to process the record ID selection. 10 and greater - View the second level message text of the next message issued (CPI432D) for an explanation of these reason codes.
Recovery Text:	To ensure an access path is considered for optimization specify that access path to be the queried file. The optimizer will first consider the access path of the file specified on the query. SQL-created indexes cannot be queried but can be deleted and recreated to increase the chance they will be considered during query optimization. The user may want to delete any access paths no longer needed.

CPI432B - Subselects processed as join query	
Message Text:	Subselects processed as join query.
Cause Text:	Two or more SQL subselects were combined together by the query optimizer and processed as a join query. Processing subselects as a join query generally results in improved performance.
Recovery Text:	None — Generally, this method of processing is a good performing option.

CPI432C - All access paths were considered for file &1	
Message Text:	All access paths were considered for file &1.
Cause Text:	The query optimizer considered all access paths built over member &3 of file &1 in library &2. The list below shows the access paths considered. If file &1 in library &2 is a logical file then the access paths specified are actually built over member &9 of physical file &7 in library &8. Following each access path name in the list is a reason code which explains how the optimizer considered the access path. &11. The reason codes and their meanings follow: 0 - The access path was used to implement the query. 1 - Access path was not in a valid state. The system invalidated the access path. 2 - Access path was not in a valid state. The user requested that the access path be rebuilt. 3 - Access path is a temporary access path (resides in library QTEMP) and was not specified as the file to be queried. 4 - The cost to use this access path, as determined by the optimizer, was higher than the cost associated with the chosen access method. 5 - The keys of the access path did not match the fields specified for the ordering/ grouping criteria. For distributed file queries, the access path keys must exactly match the ordering fields if the access path is to be used when ALWCPYDTA(*YES or *NO) is specified. 6 - The keys of the access path did not match the fields specified for the join criteria. 7 - Use of this access path would not minimize delays when reading records from the file. The user requested to minimize delays when reading records from the file. 8 - The access path cannot be used for a secondary file of the join query because it contains static select/omit selection criteria. The join-type of the query does not allow the use of select/omit access paths for secondary files. 9 - File &1 contains record ID selection. The join-type of the query forces a temporary access path to be built to process the record ID selection. 10 and greater - View the second level message text of the next message issued (CPI432D) for an explanation of these reason codes.
Recovery Text:	The user may want to delete any access paths no longer needed.

CPI432D - Additional access path reason codes were used	
Message Text:	Additional access path reason codes were used.

CPI432D - Additional access path reason codes were used	
Cause Text:	Message CPI432A or CPI432C was issued immediately before this message. Because of message length restrictions, some of the reason codes used by messages CPI432A and CPI432C are explained below rather than in those messages. The reason codes and their meanings follow: 10 - The user specified ignore decimal data errors on the query. This disallows the use of permanent access paths. 11 - The access path contains static select/omit selection criteria which is not compatible with the selection in the query. 12 - The access path contains static select/omit selection criteria whose compatibility with the selection in the query could not be determined. Either the select/omit criteria or the query selection became too complex during compatibility processing. 13 - The access path cannot be used because it contains one or more keys which may be changed by the query during an insert or update. 14 - The access path is being deleted or is being created in an uncommitted unit of work in another process. 15 - The keys of the access path matched the fields specified for the ordering/grouping criteria. However, the sequence table associated with the access path did not match the sequence table associated with the query. 16 - The keys of the access path matched the fields specified for the join criteria. However, the sequence table associated with the access path did not match the sequence table associated with the query. 17 - The left-most key of the access path did not match any fields specified for the selection criteria. Therefore, key row positioning could not be performed, making the cost to use this access path higher than the cost associated with the chosen access method. 18 - The left-most key of the access path matched a field specified for the selection criteria. However, the sequence table associated with the access path did not match the sequence table associated with the query. Therefore, key row positioning could not be performed, making the cost to use this access path higher than the cost associated with the chosen access method. 19 - The access path cannot be used because the secondary file of the join query is a select/omit logical file. The join-type requires that the select/omit access path associated with the secondary file be used or, if dynamic, that an access path be created by the system. 99 - The access path was used to gather statistics information for the query optimizer.
Recovery Text:	See prior message CPI432A or CPI432C for more information.

Because of message length restrictions, some of the reason codes used by messages CPI432A and CPI432C are explained in the message help of CPI432D. Use the message help from this message to interpret the information returned from message CPI432A or CPI432C.

CPI432E - Selection fields mapped to different attributes	
Message Text:	Selection fields mapped to different attributes.

CPI432E - Selection fields mapped to different attributes	
Cause Text:	The data type, digits, decimal position, or length of each of the following selection fields was changed so that the field could be properly compared to the literal, host variable, or field operand associated with it. Therefore, an access path cannot be used to process that selection, since no key field has attributes that match the new attributes of the field. &1. The data type of the field may have been changed to match the comparison operand. For a numeric field, the number of total digits or fractional digits of the comparison operand may have exceeded that of the field.
Recovery Text:	You may want to change each comparison operand as follows: 1 - For a literal, change the literal value so that its attributes match the field's attributes. Normally, an attributes mismatch is caused by a numeric literal that has non-significant leading or trailing zeroes. 2 - For a host variable, either change the host variable's definition to match the field's definition or define a new host variable that matches the field's definition. 3 - For a field, change the attributes of one of the fields to match the other's attributes.

CPI432F - Access path suggestion for file &1	
Message Text:	Access path suggestion for file &1.
Cause Text:	To improve performance the query optimizer is suggesting a permanent access path be built with the key fields it is recommending. The access path will access records from member &3 of file &1 in library &2. In the list of key fields that follow, the query optimizer is recommending the first &10 key fields as primary key fields. The remaining key fields are considered secondary key fields and are listed in order of expected selectivity based on this query. Primary key fields are fields that significantly reduce the number of keys selected based on the corresponding selection predicate. Secondary key fields are fields that may or may not significantly reduce the number of keys selected. It is up to the user to determine the true selectivity of secondary key fields and to determine whether those key fields should be used when creating the access path. The query optimizer is able to perform key positioning over any combination of the primary key fields, plus one additional secondary key field. Therefore it is important that the first secondary key field be the most selective secondary key field. The query optimizer will use key selection with any remaining secondary key fields. While key selection is not as fast as key positioning it can still reduce the number of keys selected. Hence, secondary key fields that are fairly selective should be included. When building the access path all primary key fields should be specified first followed by the secondary key fields which are prioritized by selectivity. The following list contains the suggested primary and secondary key fields: &11. If file &1 in library &2 is a logical file then the access path should be built over member &9 of physical file &7 in library &8.
Recovery Text:	If this query is run frequently, you may want to create the suggested access path for performance reasons. It is possible that the query optimizer will choose not to use the access path just created. For more information, refer to "Data access methods" on page 11.

CPI4330 - &6 tasks used for parallel &10 scan of file &1	
Message Text:	&6 tasks used for parallel &10 scan of file &1.
Cause Text:	&6 is the average numbers of tasks used for a &10 scan of member &3 of file &1 in library &2. If file &1 in library &2 is a logical file, then member &9 of physical file &7 in library &8 is the actual file from which records are being selected. A file name starting with *QUERY or *N for the file indicates a temporary result file is being used. The query optimizer has calculated that the optimal number of tasks is &5 which was limited for reason code &4. The reason code definitions are: 1 - The *NBRTASKS parameter value was specified for the DEGREE parameter of the CHGQRYA CL command. 2 - The optimizer calculated the number of tasks which would use all of the central processing units (CPU). 3 - The optimizer calculated the number of tasks which can efficiently run in this job's share of the memory pool. 4 - The optimizer calculated the number of tasks which can efficiently run using the entire memory pool. 5 - The optimizer limited the number of tasks to equal the number of disk units which contain the file's data. The database manager may further limit the number of tasks used if the allocation of the file's data is not evenly distributed across disk units.
Recovery Text:	To disallow usage of parallel &10 scan, specify *NONE on the query attribute degree. A larger number of tasks might further improve performance. The following actions based on the optimizer reason code might allow the optimizer to calculate a larger number: 1 - Specify a larger number of tasks value for the DEGREE parameter of the CHGQRYA CL command. Start with a value for number of tasks which is a slightly larger than &5. 2 - Simplify the query by reducing the number of fields being mapped to the result buffer or by removing expressions. Also, try specifying a number of tasks as described by reason code 1. 3 - Specify *MAX for the query attribute DEGREE. 4 - Increase the size of the memory pool. 5 - Use the CHGPF CL command or the SQL ALTER statement to redistribute the file's data across more disk units.

CPI4331 - &6 tasks used for parallel index created over file	
Message Text:	&6 tasks used for parallel index created over file &1.

CPI4331 - &6 tasks used for parallel index created over file	
Cause Text:	&6 is the average numbers of tasks used for an index created over member &3 of file &1 in library &2. If file &1 in library &2 is a logical file, then member &9 of physical file &7 in library &8 is the actual file over which the index is being built. A file name starting with *QUERY or *N for the file indicates a temporary result file is being used. The query optimizer has calculated that the optimal number of tasks is &5 which was limited for reason code &4. The definition of reason codes are: 1 - The *NBRTASKS parameter value was specified for the DEGREE parameter of the CHGQRYA CL command. 2 - The optimizer calculated the number of tasks which would use all of the central processing units (CPU). 3 - The optimizer calculated the number of tasks which can efficiently run in this job's share of the memory pool. 4 - The optimizer calculated the number of tasks which can efficiently run using the entire memory pool. The database manager may further limit the number of tasks used for the parallel index build if either the allocation of the file's data is not evenly distributed across disk units or the system has too few disk units.
Recovery Text:	To disallow usage of parallel index build, specify *NONE on the query attribute degree. A larger number of tasks might further improve performance. The following actions based on the reason code might allow the optimizer to calculate a larger number: 1 - Specify a larger number of tasks value for the DEGREE parameter of the CHGQRYA CL command. Start with a value for number of tasks which is a slightly larger than &5 to see if a performance improvement is achieved. 2 - Simplify the query by reducing the number of fields being mapped to the result buffer or by removing expressions. Also, try specifying a number of tasks for the DEGREE parameter of the CHGQRYA CL command as described by reason code 1. 3 - Specify *MAX for the query attribute degree. 4 - Increase the size of the memory pool.

CPI4332 - &1 host variables used in query	
Message Text:	&1 host variables used in query.
Cause Text:	There were &1 host variables defined for use in the query. The values used for the host variables for this open of the query follow: &2. The host variables values displayed above may have been special values. An explanation of the special values follow: - DBCS data is displayed in hex format. - *N denotes a value of NULL. - *Z denotes a zero length string. - *L denotes a value too long to display in the replacement text. - *U denotes a value that could not be displayed.

CPI4332 - &1 host variables used in query	
Recovery Text:	None

CPI4333 - Hashing algorithm used to process join	
Message Text:	Hashing algorithm used to process join.
Cause Text:	The hash join method is typically used for longer running join queries. The original query will be subdivided into hash join steps. Each hash join step will be optimized and processed separately. Debug messages which explain the implementation of each hash join step follow this message in the joblog. The list below shows the names of the files or the table functions used in this query. If the entry is for a file, the format of the entry in this list is the number of the hash join step, the filename as specified in the query, the member name as specified in the query, the filename actually used in the hash join step, and the member name actually used in the hash join step. If the entry is for a table function, the format of the entry in this list is the number of the hash join step and the function name as specified in the query. If there are two or more files or functions listed for the same hash step, then that hash step is implemented with nested loop join.
Recovery Text:	The hash join method is usually a good implementation choice, however, if you want to disallow the use of this method specify ALWCPYDTA(*YES).

CPI4334 - Query implemented as reusable ODP	
Message Text:	Query implemented as reusable ODP.
Cause Text:	The query optimizer built the access plan for this query such that a reusable open data path (ODP) will be created. This plan will allow the query to be run repeatedly for this job without having to rebuild the ODP each time. This normally improves performance because the ODP is created only once for the job.
Recovery Text:	Generally, reusable ODPs perform better than non-reusable ODPs.

CPI4335 - Optimizer debug messages for hash join step &1 follow	
Message Text:	Optimizer debug messages for hash join step &1 follow:
Cause Text:	This join query is implemented using the hash join algorithm. The optimizer debug messages that follow provide the query optimization information about hash join step &1.
Recovery Text:	Refer to "Data access methods" on page 11 for more information about hashing algorithm for join processing.

CPI4336 - Group processing generated	
Message Text:	Group processing generated.
Cause Text:	Group processing (GROUP BY) was added to the query step. Adding the group processing reduced the number of result records which should, in turn, improve the performance of subsequent steps.

CPI4336 - Group processing generated	
Recovery Text:	For more information refer to “Data access methods” on page 11

CPI4337 - Temporary hash table build for hash join step &1	
Message Text:	Temporary hash table built for hash join step &1.
Cause Text:	A temporary hash table was created to contain the results of hash join step &1. This process took &2 minutes and &3 seconds. The temporary hash table created contains &4 records. The total size of the temporary hash table in units of 1024 bytes is &5. A list of the fields which define the hash keys follow:
Recovery Text:	Refer to “Data access methods” on page 11 for more information about hashing algorithm for join processing.

CPI4338 - &1 Access path(s) used for bitmap processing of file &2	
Message Text:	&1 Access path(s) used for bitmap processing of file &2.
Cause Text:	Bitmap processing was used to access records from member &4 of file &2 in library &3. Bitmap processing is a method of allowing one or more access path(s) to be used to access the selected records from a file. Using bitmap processing, record selection is applied against each access path, similar to key row positioning, to create a bitmap. The bitmap has marked in it only the records of the file that are to be selected. If more than one access path is used, the resulting bitmaps are merged together using boolean logic. The resulting bitmap is then used to reduce access to just those records actually selected from the file. Bitmap processing is used in conjunction with the two primary access methods: arrival sequence (CPI4327 or CPI4329) or keyed access (CPI4326 or CPI4328). The message that describes the primary access method immediately precedes this message. When the bitmap is used with the keyed access method then it is used to further reduce the number of records selected by the primary access path before retrieving the selected records from the file. When the bitmap is used with arrival sequence then it allows the sequential scan of the file to skip records which are not selected by the bitmap. This is called skip sequential processing. The list below shows the names of the access paths used in the bitmap processing: &8 If file &2 in library &3 is a logical file then member &7 of physical file &5 in library &6 is the actual file being accessed.
Recovery Text:	Refer to “Data access methods” on page 11 for more information about bitmap processing.

CPI433A - Unable to retrieve query options file	
Message Text:	Unable to retrieve query options file.

CPI433A - Unable to retrieve query options file	
Cause Text:	Unable to retrieve the query options from member &3 in file &2 in library &1 for reason code &4. The reason codes and their meanings follow: 1 - Library &1 was not found. 2 - File &2 in library &1 was not found. 3 - The file was damaged. 4 - The file was locked by another process which prevented successful retrieval of the query options. 5 - File &2 and the internal query options structures are out of sync. 6 - An unexpected error occurred while trying to retrieve the options file. The query options file is used by the Query Optimizer to determine how a query will be implemented.
Recovery Text:	Default query options will be used, unless one of the following actions are taken, based on the reason code above. 1 - Either create the library (CRTLIB command) or correct the library name and then try the request again. 2 - Either specify the library name that contains the query options file or create a duplicate object (CRTDUPOBJ command) of file &2 from library QSYS into the specified library. 4 - Wait for lock on file &2 in library &1 to be released and try the request again. 3, 5, or 6 - Delete query options file &2 in library &1 and then duplicate it from QSYS. If the problem still persists, report the problem (ANZPRB command).

CPI433B - Unable to update query options file	
Message Text:	Unable to update query options file.
Cause Text:	An error occurred while trying to update the query options from member &3, file &2, library &1 for reason code &4. The reason codes and their meanings follow: 1 - The library &1 was not found. 2 - The file &2 in library &1 was not found. 3 - The parameter &5 was not found. 4 - The value &6 for parameter &5 was not valid. 5 - An unexpected error occurred while trying to update the options file.
Recovery Text:	Do one of the following actions based on the reason code above. 1 - Either create the library (CRTLIB) command or correct the library name and then try the request again. 2 - Either specify the library name that contains the query options file or create duplicate object (CRTDUPOBJ) command of QAQQINI from library QSYS into the specified library. 3 - Either specify a valid parameter or correct the parameter name and then try the request again. 4 - Either specify a valid parameter value or correct the parameter value and then try the request again. (WRKJOB) command.

CPI433C - Library &1 not found	
Message Text:	Library &1 not found.
Cause Text:	The specified library does not exist, or the name of the library is not spelled correctly.
Recovery Text:	Correct the spelling of the library name, or specify the name of an existing library. Then try the request again.

CPI433D - Query options used to build the query access plan	
Message Text:	Query options used to build the query access plan.
Cause Text:	The access plan that was saved was created with query options retrieved from file &2 in library &1.
Recovery Text:	None

CPI433E - User-defined function &4 found in library &1	
Message Text:	User-defined function &4 found in library &1.
Cause Text:	Function &4 was resolved to library &1. The specific name of the function is &5. If the function is defined to use an external program, the associated program or service program is &3 in library &2.
Recovery Text:	Refer to the SQL programming topic collection, for more information on user-defined functions.

CPI433F - Multiple join classes used to process join	
Message Text:	Multiple join classes used to process join.
Cause Text:	Multiple join classes are used when join queries are written that have conflicting operations or cannot be implemented as a single query. Each join class step will be optimized and processed separately. Debug messages detailing the implementation of each join class follow this message in the joblog. The list below shows the file names of the files used in this query. The format of each entry in this list is the number of the join class step, the number of the join position in the join class step, the file name as specified in the query, the member name as specified in the query, the file name actually used in the join class step, and the member name actually used in the join class step.
Recovery Text:	Refer to “Join optimization” on page 59 for more information about join classes.

CPI4340 - Optimizer debug messages for join class step &1 follow	
Message Text:	Optimizer debug messages for join class step &1 follow:
Cause Text:	This join query is implemented using multiple join classes. The optimizer debug messages that follow provide the query optimization information about join class step &1.

CPI4340 - Optimizer debug messages for join class step &1 follow	
Recovery Text:	Refer to “Join optimization” on page 59 for more information about join classes.

CPI4341 - Performing distributed query	
Message Text:	Performing distributed query.
Cause Text:	Query contains a distributed file. The query was processed in parallel on the following nodes: &1.
Recovery Text:	For more information about processing of distributed files, refer to the Distributed database programming topic collection.

CPI4342 - Performing distributed join for query	
Message Text:	Performing distributed join for query.
Cause Text:	Query contains join criteria over a distributed file and a distributed join was performed, in parallel, on the following nodes: &1. The library, file and member names of each file involved in the join follow: &2. A file name beginning with *QQTDF indicates it is a temporary distributed result file created by the query optimizer and it will not contain an associated library or member name.
Recovery Text:	For more information about processing of distributed files, refer to the Distributed database programming.

CPI4343 - Optimizer debug messages for distributed query step &1 of &2 follow	
Message Text:	Optimizer debug messages for distributed query step &1 of &2 follow:
Cause Text:	A distributed file was specified in the query which caused the query to be processed in multiple steps. The optimizer debug messages that follow provide the query optimization information about distributed step &1 of &2 total steps.
Recovery Text:	For more information about processing of distributed files, refer to the Distributed database programming.

CPI4345 - Temporary distributed result file &3 built for query	
Message Text:	Temporary distributed result file &3 built for query.

CPI4345 - Temporary distributed result file &3 built for query	
Cause Text:	Temporary distributed result file &3 was created to contain the intermediate results of the query for reason code &6. The reason codes and their meanings follow: 1 - Data from member &2 of &7 &8 in library &1 was directed to other nodes. 2 - Data from member &2 of &7 &8 in library &1 was broadcast to all nodes. 3 - Either the query contains grouping fields (GROUP BY) that do not match the partitioning keys of the distributed file or the query contains grouping criteria but no grouping fields were specified or the query contains a subquery. 4 - Query contains join criteria over a distributed file and the query was processed in multiple steps. A library and member name of *N indicates the data comes from a query temporary distributed file. File &3 was built on nodes: &9. It was built using partitioning keys: &10. A partitioning key of *N indicates no partitioning keys were used when building the temporary distributed result file.
Recovery Text:	If the reason code is: 1 - Generally, a file is directed when the join fields do not match the partitioning keys of the distributed file. When a file is directed, the query is processed in multiple steps and processed in parallel. A temporary distributed result file is required to contain the intermediate results for each step. 2 - Generally, a file is broadcast when join fields do not match the partitioning keys of either file being joined or the join operator is not an equal operator. When a file is broadcast the query is processed in multiple steps and processed in parallel. A temporary distributed result file is required to contain the intermediate results for each step. 3 - Better performance may be achieved if grouping fields are specified that match the partitioning keys. 4 - Because the query is processed in multiple steps, a temporary distributed result file is required to contain the intermediate results for each step. See preceding message CPI4342 to determine which files were joined together. For more information about processing of distributed files, refer to the Distributed database programming

CPI4346 - Optimizer debug messages for query join step &1 of &2 follow	
Message Text:	Optimizer debug messages for query join step &1 of &2 follow:
Cause Text:	Query processed in multiple steps. The optimizer debug messages that follow provide the query optimization information about join step &1 of &2 total steps.
Recovery Text:	No recovery necessary.

CPI4347 - Query being processed in multiple steps	
Message Text:	Query being processed in multiple steps.

CPI4347 - Query being processed in multiple steps	
Cause Text:	The original query will be subdivided into multiple steps. Each step will be optimized and processed separately. Debug messages which explain the implementation of each step follow this message in the joblog. The list below shows the file names of the files used in this query. The format of each entry in this list is the number of the join step, the filename as specified in the query, the member name as specified in the query, the filename actually used in the step, and the member name actually used in the step.
Recovery Text:	No recovery necessary.

CPI4348 - The ODP associated with the cursor was hard closed	
Message Text:	The ODP associated with the cursor was hard closed.
Cause Text:	The Open Data Path (ODP) for this statement or cursor has been hard closed for reason code &1. The reason codes and their meanings follow: 1 - Either the length of the new LIKE pattern is zero and the length of the old LIKE pattern is nonzero or the length of the new LIKE pattern is nonzero and the length of the old LIKE pattern is zero. 2 - An additional wildcard was specified in the LIKE pattern on this invocation of the cursor. 3 - SQL indicated to the query optimizer that the cursor cannot be refreshed. 4 - The system code could not obtain a lock on the file being queried. 5 - The length of the host variable value is too large for the the host variable as determined by the query optimizer. 6 - The size of the ODP to be refreshed is too large. 7 - Refresh of the local ODP of a distributed query failed. 8 - SQL hard closed the cursor prior to the fast path refresh code.
Recovery Text:	In order for the cursor to be used in a reusable mode, the cursor cannot be hard closed. Look at the reason why the cursor was hard closed and take the appropriate actions to prevent a hard close from occurring.

CPI4349 - Fast past refresh of the host variables values is not possible	
Message Text:	Fast past refresh of the host variable values is not possible.

CPI4349 - Fast past refresh of the host variables values is not possible	
Cause Text:	The Open Data Path (ODP) for this statement or cursor could not invoke the fast past refresh code for reason code &1. The reason codes and their meanings follow: 1 - The new host variable value is not null and old host variable value is null or the new host variable value is zero length and the old host variable value is not zero length. 2 - The attributes of the new host variable value are not the same as the attributes of the old host variable value. 3 - The length of the host variable value is either too long or too short. The length difference cannot be handled in the fast path refresh code. 4 - The host variable has a data type of IGC ONLY and the the length is not even or is less than 2 bytes. 5 - The host variable has a data type of IGC ONLY and the new host variable value does not contain an even number of bytes. 6 - A translate table with substitution characters was used. 7 - The host variable contains DBCS data and a CCSID translate table with substitution characters is required. 8 - The host variable contains DBCS that is not well formed. That is, a shift-in without a shift-out or visa versa. 9 - The host variable must be translated with a sort sequence table and the sort sequence table contains substitution characters. 10 - The host variable contains DBCS data and must be translated with a sort sequence table that contains substitution characters. 11 - The host variable is a Date, Time or Timestamp data type and the length of the host variable value is either too long or too short.
Recovery Text:	Look at the reason why fast path refresh could not be used and take the appropriate actions so that fast path refresh can be used on the next invocation of this statement or cursor.

CPI434 - Member &3 was opened with fewer open options than were specified	
Message Text:	Member &3 was opened with fewer open options than were specified.
Cause Text:	An INSTEAD OF trigger is being used for some of the open options. However there is an additional INSTEAD OF trigger on an underlying SQL view file whose trigger actions cannot be used. An open request can support INSTEAD OF triggers from only one SQL view file. The member could not be opened with the following open options: &4.
Recovery Text:	When adding an INSTEAD OF trigger, specify trigger actions for all of the requested open options.

CPI434E - Query could not be run using SQE	
Message Text:	Query could not be run using SQE.
Cause Text:	The query was run using CQE (Current Query Engine). The query could not be run using SQE (SQL Query Engine) for reason code &1. The reason codes and their meanings follow: 1 -- Sort sequence table &2 in library &3 is an ICU (International Components of Unicode) sort sequence table that is not supported by SQE.

CPI434E - Query could not be run using SQE	
Recovery Text:	Recovery for reason code 1: To run the query using SQE, specify a version of the ICU sort sequence table that is &4 or later.

CPI4350 - Materialized query tables were considered for optimization	
Message Text:	Materialized query tables were considered for optimization.

CPI4350 - Materialized query tables were considered for optimization	
Cause Text:	The query optimizer considered usage of materialized query tables for this query. Following each materialized query table name in the list is a reaon code which explains why the materialized query table was not used. A reason code of 0 indicates that the materialized query table was used to implement the query. The reason codes and their meanings follow: 1 - The cost to use the materialized query table, as determined by the optimizer, was higher than the cost associated with the chosen implementation. 2 - The join specified in the materialized query was not compatible with the query. 3 - The materialized query table had predicates that were not matched in the query. 4 - The grouping or distinct specified in the materialized query table is not compatible with the grouping or distinct specified in the query. 5 - The query specified columns that were not in the select-list of the materialized query table. 6 - The materialized query table query contains functionality that is not supported by the query optimizer. 7 - The materialized query table specified the DISABLE QUERY OPTIMIZATION clause. 8 - The ordering specified in the materialized query table is not compatible with the ordering specified in the query. 9 - The query contains functionality that is not supported by the materialized query table matching algorithm. 10 - Materialized query tables may not be used for this query. 11 - The refresh age of this materialized query table exceeds the duration specified by the MATERIALIZED_QUERY_TABLE_REFRESH_AGE QAQQINI option. 12 - The commit level of the materialized query table is lower than the commit level specified for the query. 14 - The FETCH FOR FIRST n ROWS clause of the materialized query table is not compatible with the query. 15 - The QAQQINI options used to create the materialized query table are not compatible with the QAQQINI options used to run this query. 16 - The materialized query table is not usable. 17 - The UNION specified in the materialized query table is not compatible with the query. 18 - The constants specified in the materialized query table are not compatible with host variable values specified in the query. 19 - The materialized query table is in Check Pending status and cannot be used. 20 - The UDTF specified in the materialized query table is not compatible with UDTF in the query. 21 - The Values clause specified in the materialized query table is not compatible with Values specified in the query.
Recovery Text:	The user may want to delete any materialized query tables that are no longer needed.

CPI4351 - Additional reason codes for query access plan has been rebuilt	
Message Text:	Additional reason codes for query access plan has been rebuilt.
Cause Text:	Message CPI4323 was issued immediately before this message. Because of message length restrictions, some of the reason codes used by message CPI4323 are explained below rather than in that message. The CPI4323 message was issued for reason code &13. The additional reason codes and their meaning follow: 20 - Referential or check constraints for member &19 of file &17 in library &18 have changed since the access plan was generated. 21 - Materialized query tables for member &22 of file &20 in library &21 have changed since the access plan was generated. If the file is *N then the file name is not available. 22 - The value of a host variable changed and the access plan is no longer valid. 23 - Adaptive Query Processing (AQP) determined that a new access plan is needed.
Recovery Text:	See the prior message CPI4323 for more information.

CPI436A - Database monitor started for job &1, monitor ID &2	
Message Text:	Database monitor started for job &1, monitor ID &2.
Cause Text:	The database monitor was started for job &1. The system generated monitor ID for this database monitor is &2. If multiple monitors have been started using the same generic job name, the monitor ID is needed to uniquely identify which monitor is to be ended with the ENDDBMON command.
Recovery Text:	If multiple monitors have been started using the same generic job name, remember the monitor ID. The monitor ID will be required when using the ENDDBMON command to end this specific monitor.

Query optimization performance information messages and open data paths

Several of the following SQL runtime messages refer to open data paths.

An open data path (ODP) definition is an internal object that is created when a cursor is opened or when other SQL statements are run. It provides a direct link to the data so that I/O operations can occur. ODPs are used on OPEN, INSERT, UPDATE, DELETE, and SELECT INTO statements to perform their respective operations on the data.

Even though SQL cursors are closed and SQL statements have run, in many cases, the database manager saves the associated ODPs of the SQL operations. These ODPs are then reused the next time the statement is run. For example, an SQL CLOSE statement could close the SQL cursor, but leave the ODP available to use again the next time the cursor is opened. This technique can significantly reduce the processing and response time in running SQL statements.

The ability to reuse ODPs when SQL statements are run repeatedly is an important consideration in achieving faster performance.

SQL7910 - All SQL cursors closed	
Message Text:	SQL cursors closed.

SQL7910 - All SQL cursors closed	
Cause Text:	SQL cursors have been closed and all Open Data Paths (ODPs) have been deleted, except those that were opened by programs with the CLOSQLCSR(*ENDJOB) option or were opened by modules with the CLOSQLCSR(*ENDACTGRP) option. All SQL programs on the call stack have completed, and the SQL environment has been exited. This process includes the closing of cursors, the deletion of ODPs, the removal of prepared statements, and the release of locks.
Recovery Text:	To keep cursors, ODPs, prepared statements, and locks available after the completion of a program, use the CLOSQLCSR precompile parameter. -- The *ENDJOB option will allow the user to keep the SQL resources active for the duration of the job. -- The *ENDSQL option will allow the user to keep SQL resources active across program calls, provided the SQL environment stays resident. Running an SQL statement in the first program of an application will keep the SQL environment active for the duration of that application. -- The *ENDPGM option, which is the default for non-Integrated Language Environment® (ILE) programs, causes all SQL resources to only be accessible by the same invocation of a program. Once an *ENDPGM program has completed, if it is called again, the SQL resources are no longer active. -- The *ENDMOD option causes all SQL resources to only be accessible by the same invocation of the module. -- The *ENDACTGRP option, which is the default for ILE modules, will allow the user to keep the SQL resources active for the duration of the activation group.

SQL7911 - ODP reused	
Message Text:	ODP reused.
Cause Text:	An ODP that was previously created has been reused. There was a reusable Open Data Path (ODP) found for this SQL statement, and it has been used. The reusable ODP may have been from the same call to a program or a previous call to the program. A reuse of an ODP will not generate an OPEN entry in the journal.
Recovery Text:	None

SQL7912 - ODP created	
Message Text:	ODP created.
Cause Text:	An Open Data Path (ODP) has been created. No reusable ODP could be found. This occurs in the following cases: -- This is the first time the statement has been run. -- A RCLRSC has been issued since the last run of this statement. -- The last run of the statement caused the ODP to be deleted. -- If this is an OPEN statement, the last CLOSE of this cursor caused the ODP to be deleted. -- The Application Server (AS) has been changed by a CONNECT statement.

SQL7912 - ODP created	
Recovery Text:	If a cursor is being opened many times in an application, it is more efficient to use a reusable ODP, and not create an ODP every time. This also applies to repeated runs of INSERT, UPDATE, DELETE, and SELECT INTO statements. If ODPs are being created on every open, see the close message to determine why the ODP is being deleted.

The first time that the statement is run or the cursor is opened for a process, an ODP must always be created. However, if this message appears on every statement run or cursor open, use the tips recommended in "Retaining cursor positions for non-ILE program calls" on page 249 in your application.

SQL7913 - ODP deleted	
Message Text:	ODP deleted.
Cause Text:	The Open Data Path (ODP) for this statement or cursor has been deleted. The ODP was not reusable. This could be caused by using a host variable in a LIKE clause, ordering on a host variable, or because the query optimizer chose to accomplish the query with an ODP that was not reusable.
Recovery Text:	See previous query optimizer messages to determine how the cursor was opened.

SQL7914 - ODP not deleted	
Message Text:	ODP not deleted.
Cause Text:	The Open Data Path (ODP) for this statement or cursor has not been deleted. This ODP can be reused on a subsequent run of the statement. This will not generate an entry in the journal.
Recovery Text:	None

SQL7915 - Access plan for SQL statement has been built	
Message Text:	Access plan for SQL statement has been built.
Cause Text:	SQL had to build the access plan for this statement at run time. This occurs in the following cases: -- The program has been restored from a different release and this is the first time this statement has been run. -- All the files required for the statement did not exist at precompile time, and this is the first time this statement has been run. -- The program was precompiled using SQL naming mode, and the program owner has changed since the last time the program was called.
Recovery Text:	This is normal processing for SQL. Once the access plan is built, it will be used on subsequent runs of the statement.

SQL7916 - Blocking used for query	
Message Text:	Blocking used for query.

SQL7916 - Blocking used for query	
Cause Text:	Blocking has been used in the implementation of this query. SQL will retrieve a block of records from the database manager on the first FETCH statement. Additional FETCH statements have to be issued by the calling program, but they do not require SQL to request more records, and therefore will run faster.
Recovery Text:	SQL attempts to utilize blocking whenever possible. In cases where the cursor is not update capable, and commitment control is not active, there is a possibility that blocking will be used.

SQL7917 - Access plan not updated	
Message Text:	Access plan not updated.
Cause Text:	The query optimizer rebuilt the access plan for this statement, but the program could not be updated. Another job may be running the program. The program cannot be updated with the new access plan until a job can obtain an exclusive lock on the program. The exclusive lock cannot be obtained if another job is running the program, if the job does not have proper authority to the program, or if the program is currently being saved. The query will still run, but access plan rebuilds will continue to occur until the program is updated.
Recovery Text:	See previous messages from the query optimizer to determine why the access plan has been rebuilt. To ensure that the program gets updated with the new access plan, run the program when no other active jobs are using it.

SQL7918 - Reusable ODP deleted	
Message Text:	Reusable ODP deleted. Reason code &1.

SQL7918 - Reusable ODP deleted	
Cause Text:	An existing Open Data Path (ODP) was found for this statement, but it could not be reused for reason &1. The statement now refers to different files or uses different override options than are in the ODP. Reason codes and their meanings are: 1 -- Commitment control isolation level is not compatible. 2 -- The statement contains SQL special register USER, CURRENT DEBUG MODE, CURRENT DECFLOAT ROUNDING MODE, or CURRENT TIMEZONE, and the value for one of these registers has changed. 3 -- The PATH used to locate an SQL function has changed. 4 -- The job default CCSID has changed. 5 -- The library list has changed, such that a file is found in a different library. This only affects statements with unqualified table names, when the table exists in multiple libraries. 6 -- The file, library, or member for the original ODP was changed with an override. 7 -- An OVRDBF or DLTOVR command has been issued. A file referred to in the statement now refers to a different file, library, or member. 8 -- An OVRDBF or DLTOVR command has been issued, causing different override options, such as different SEQONLY or WAITRCD values. 9 -- An error occurred when attempting to verify the statement override information is compatible with the reusable ODP information. 10 -- The query optimizer has determined the ODP cannot be reused. 11 -- The client application requested not to reuse ODPs.
Recovery Text:	Do not change the library list, the override environment, or the values of the special registers if reusable ODPs are to be used.

SQL7919 - Data conversion required on FETCH or embedded SELECT	
Message Text:	Data conversion required on FETCH or embedded SELECT.

SQL7919 - Data conversion required on FETCH or embedded SELECT	
Cause Text:	Host variable &2 requires conversion. The data retrieved for the FETCH or embedded SELECT statement can not be directly moved to the host variables. The statement ran correctly. Performance, however, would be improved if no data conversion was required. The host variable requires conversion for reason &1. -- Reason 1 - host variable &2 is a character or graphic string of a different length than the value being retrieved. -- Reason 2 - host variable &2 is a numeric type that is different than the type of the value being retrieved. -- Reason 3 - host variable &2 is a C character or C graphic string that is NUL-terminated, the program was compiled with option *CNULRQD specified, and the statement is a multiple-row FETCH. -- Reason 4 - host variable &2 is a variable length string and the value being retrieved is not. -- Reason 5 - host variable &2 is not a variable length string and the value being retrieved is. -- Reason 6 - host variable &2 is a variable length string whose maximum length is different than the maximum length of the variable length value being retrieved. -- Reason 7 - a data conversion was required on the mapping of the value being retrieved to host variable &2, such as a CCSID conversion. -- Reason 8 - a DRDA connection was used to get the value being retrieved into host variable &2. The value being retrieved is either null capable or varying-length, is contained in a partial row, or is a derived expression. -- Reason 10 - the length of host variable &2 is too short to hold a TIME or TIMESTAMP value being retrieved. -- Reason 11 - host variable &2 is of type DATE, TIME or TIMESTAMP, and the value being retrieved is a character string. -- Reason 12 - too many host variables were specified and records are blocked. Host variable &2 does not have a corresponding column returned from the query. -- Reason 13 - a DRDA connection was used for a blocked FETCH and the number of host variables specified in the INTO clause is less than the number of result values in the select list. -- Reason 14 - a LOB Locator was used and the commitment control level of the process was not *ALL.
Recovery Text:	To get better performance, attempt to use host variables of the same type and length as their corresponding result columns.

SQL7939 - Data conversion required on INSERT or UPDATE	
Message Text:	Data conversion required on INSERT or UPDATE.

SQL7939 - Data conversion required on INSERT or UPDATE	
Cause Text:	The INSERT or UPDATE values can not be directly moved to the columns because the data type or length of a value is different than one of the columns. The INSERT or UPDATE statement ran correctly. Performance, however, would be improved if no data conversion was required. The reason data conversion is required is &1. -- Reason 1 is that the INSERT or UPDATE value is a character or graphic string of a different length than column &2. -- Reason 2 is that the INSERT or UPDATE value is a numeric type that is different than the type of column &2. -- Reason 3 is that the INSERT or UPDATE value is a variable length string and column &2 is not. -- Reason 4 is that the INSERT or UPDATE value is not a variable length string and column &2 is. -- Reason 5 is that the INSERT or UPDATE value is a variable length string whose maximum length is different that the maximum length of column &2. -- Reason 6 is that a data conversion was required on the mapping of the INSERT or UPDATE value to column &2, such as a CCSID conversion. -- Reason 7 is that the INSERT or UPDATE value is a character string and column &2 is of type DATE, TIME, or TIMESTAMP. -- Reason 8 is that the target table of the INSERT is not a SQL table.
Recovery Text:	To get better performance, try to use values of the same type and length as their corresponding columns.

PRTSQLINF message reference

The following messages are returned from **PRTSQLINF**.

SQL400A - Temporary distributed result file &1 was created to contain join result	
Message Text:	Temporary distributed result file &1 was created to contain join result. Result file was directed.
Cause Text:	Query contains join criteria over a distributed file and a distributed join was performed in parallel. A temporary distributed result file was created to contain the results of the distributed join.
Recovery Text:	For more information about processing of distributed files, refer to the Distributed database programming topic collection.

SQL400B - Temporary distributed result file &1 was created to contain join result	
Message Text:	Temporary distributed result file &1 was created to contain join result. Result file was broadcast.
Cause Text:	Query contains join criteria over a distributed file and a distributed join was performed in parallel. A temporary distributed result file was created to contain the results of the distributed join.
Recovery Text:	For more information about processing of distributed files, refer to the Distributed database programming topic collection.

SQL400C - Optimizer debug messages for distributed query step &1 and &2 follow	
Message Text:	Optimizer debug messages for distributed query step &1 of &2 follow:
Cause Text:	A distributed file was specified in the query which caused the query to be processed in multiple steps. The optimizer debug messages that follow provide the query optimization information about the current step.
Recovery Text:	For more information about processing of distributed files, refer to the Distributed database programming topic collection.

SQL400D - GROUP BY processing generated	
Message Text:	GROUP BY processing generated.
Cause Text:	GROUP BY processing was added to the query step. Adding the GROUP BY reduced the number of result rows which should, in turn, improve the performance of subsequent steps.
Recovery Text:	For more information refer to the SQL programming topic collection.

SQL400E - Temporary distributed result file &1 was created while processing distributed subquery	
Message Text:	Temporary distributed result file &1 was created while processing distributed subquery.
Cause Text:	A temporary distributed result file was created to contain the intermediate results of the query. The query contains a subquery which requires an intermediate result.
Recovery Text:	Generally, if the fields correlated between the query and subquery do not match the partition keys of the respective files, the query must be processed in multiple steps and a temporary distributed file will be built to contain the intermediate results. For more information about processing of distributed files, refer to the Distributed database programming topic collection.

SQL4001 - Temporary result created	
Message Text:	Temporary result created.
Cause Text:	Conditions exist in the query which cause a temporary result to be created. One of the following reasons may be the cause for the temporary result: -- The table is a join logical file and its join type (JDFTVAL) does not match the join-type specified in the query. -- The format specified for the logical file refers to more than one physical table. -- The table is a complex SQL view requiring a temporary table to contain the results of the SQL view. -- The query contains grouping columns (GROUP BY) from more than one table, or contains grouping columns from a secondary table of a join query that cannot be reordered.
Recovery Text:	Performance may be improved if the query can be changed to avoid temporary results.

SQL4002 - Reusable ODP sort used	
Message Text:	Reusable ODP sort used.
Cause Text:	Conditions exist in the query which cause a sort to be used. This allowed the open data path (ODP) to be reusable. One of the following reasons may be the cause for the sort: -- The query contains ordering columns (ORDER BY) from more than one table, or contains ordering columns from a secondary table of a join query that cannot be reordered. -- The grouping and ordering columns are not compatible. -- DISTINCT was specified for the query. -- UNION was specified for the query. -- The query had to be implemented using a sort. Key length of more than 2000 bytes, more than 120 ordering columns, or an ordering column containing a reference to an external user-defined function was specified for ordering. -- The query optimizer chose to use a sort rather than an index to order the results of the query.
Recovery Text:	A reusable ODP generally results in improved performance when compared to a non-reusable ODP.

SQL4003 - UNION	
Message Text:	UNION, EXCEPT, or INTERSECT.
Cause Text:	A UNION, EXCEPT, or INTERSECT operator was specified in the query. The messages preceding this keyword delimiter correspond to the subselect preceding the UNION, EXCEPT, or INTERSECT operator. The messages following this keyword delimiter correspond to the subselect following the UNION, EXCEPT, or INTERSECT operator.
Recovery Text:	None

SQL4004 - SUBQUERY	
Message Text:	SUBQUERY.
Cause Text:	The SQL statement contains a subquery. The messages preceding the SUBQUERY delimiter correspond to the subselect containing the subquery. The messages following the SUBQUERY delimiter correspond to the subquery.
Recovery Text:	None

SQL4005 - Query optimizer timed out for table &1	
Message Text:	Query optimizer timed out for table &1.
Cause Text:	The query optimizer timed out before it could consider all indexes built over the table. This is not an error condition. The query optimizer may time out in order to minimize optimization time. The query can be run in debug mode (STRDBG) to see the list of indexes which were considered during optimization. The table number refers to the relative position of this table in the query.

SQL4005 - Query optimizer timed out for table &1	
Recovery Text:	To ensure an index is considered for optimization, specify the logical file of the index as the table to be queried. The optimizer will first consider the index of the logical file specified on the SQL select statement. Note that SQL created indexes cannot be queried. An SQL index can be deleted and recreated to increase the chances it will be considered during query optimization. Consider deleting any indexes no longer needed.

SQL4006 - All indexes considered for table &1	
Message Text:	All indexes considered for table &1.
Cause Text:	The query optimizer considered all index built over the table when optimizing the query. The query can be run in debug mode (STRDBG) to see the list of indexes which were considered during optimization. The table number refers to the relative position of this table in the query.
Recovery Text:	None

SQL4007 - Query implementation for join position &1 table &2	
Message Text:	Query implementation for join position &1 table &2.
Cause Text:	The join position identifies the order in which the tables are joined. A join position of 1 indicates this table is the first, or left-most, table in the join order. The table number refers to the relative position of this table in the query.
Recovery Text:	Join order can be influenced by adding an ORDER BY clause to the query. Refer to "Join optimization" on page 59 for more information about join optimization and tips to influence join order.

SQL4008 - Index &1 used for table &2	
Message Text:	Index &1 used for table &2.
Cause Text:	The index was used to access rows from the table for one of the following reasons: -- Row selection. -- Join criteria. -- Ordering/grouping criteria. -- Row selection and ordering/grouping criteria. The table number refers to the relative position of this table in the query. The query can be run in debug mode (STRDBG) to determine the specific reason the index was used.
Recovery Text:	None

SQL4009 - Index created for table &1	
Message Text:	Index created for table &1.

SQL4009 - Index created for table &1	
Cause Text:	A temporary index was built to access rows from the table for one of the following reasons: -- Perform specified ordering/grouping criteria. -- Perform specified join criteria. The table number refers to the relative position of this table in the query.
Recovery Text:	To improve performance, consider creating a permanent index if the query is run frequently. The query can be run in debug mode (STRDBG) to determine the specific reason the index was created and the key columns used when creating the index. NOTE: If permanent index is created, it is possible the query optimizer may still choose to create a temporary index to access the rows from the table.

SQL401A - Processing grouping criteria for query containing a distributed table	
Message Text:	Processing grouping criteria for query containing a distributed table.
Cause Text:	Grouping for queries that contain distributed tables can be implemented using either a one or two step method. If the one step method is used, the grouping columns (GROUP BY) match the partitioning keys of the distributed table. If the two step method is used, the grouping columns do not match the partitioning keys of the distributed table or the query contains grouping criteria but no grouping columns were specified. If the two step method is used, message SQL401B will appear followed by another SQL401A message.
Recovery Text:	For more information about processing of distributed tables, refer to the Distributed database programming topic collection.

SQL401B - Temporary distributed result table &1 was created while processing grouping criteria	
Message Text:	Temporary distributed result table &1 was created while processing grouping criteria.
Cause Text:	A temporary distributed result table was created to contain the intermediate results of the query. Either the query contains grouping columns (GROUP BY) that do not match the partitioning keys of the distributed table or the query contains grouping criteria but no grouping columns were specified.
Recovery Text:	For more information about processing of distributed tables, refer to the Distributed database programming topic collection.

SQL401C - Performing distributed join for query	
Message Text:	Performing distributed join for query.
Cause Text:	Query contains join criteria over a distributed table and a distributed join was performed in parallel. See the following SQL401F messages to determine which tables were joined together.
Recovery Text:	For more information about processing of distributed tables, refer to the Distributed database programming topic collection.

SQL401D - Temporary distributed result table &1 was created because table &2 was directed	
Message Text:	Temporary distributed result table &1 was created because table &2 was directed.

SQL401D - Temporary distributed result table &1 was created because table &2 was directed	
Cause Text:	Temporary distributed result table was created to contain the intermediate results of the query. Data from a distributed table in the query was directed to other nodes.
Recovery Text:	Generally, a table is directed when the join columns do not match the partitioning keys of the distributed table. When a table is directed, the query is processed in multiple steps and processed in parallel. A temporary distributed result file is required to contain the intermediate results for each step. For more information about processing of distributed tables, refer to the Distributed database programming topic collection.

SQL401E - Temporary distributed result table &1 was created because table &2 was broadcast	
Message Text:	Temporary distributed result table &1 was created because table &2 was broadcast.
Cause Text:	Temporary distributed result table was created to contain the intermediate results of the query. Data from a distributed table in the query was broadcast to all nodes.
Recovery Text:	Generally, a table is broadcast when join columns do not match the partitioning keys of either table being joined or the join operator is not an equal operator. When a table is broadcast the query is processed in multiple steps and processed in parallel. A temporary distributed result table is required to contain the intermediate results for each step. For more information about processing of distributed tables, refer to the Distributed database programming topic collection.

SQL401F - Table &1 used in distributed join	
Message Text:	Table &1 used in distributed join.
Cause Text:	Query contains join criteria over a distributed table and a distributed join was performed in parallel.
Recovery Text:	For more information about processing of distributed tables, refer to the Distributed database programming topic collection.

SQL4010 - Table scan access for table &1	
Message Text:	Table scan access for table &1.
Cause Text:	Table scan access was used to select rows from the table. The table number refers to the relative position of this table in the query.
Recovery Text:	Table scan is generally a good performing option when selecting a high percentage of rows from the table. The use of an index, however, may improve the performance of the query when selecting a low percentage of rows from the table.

SQL4011 - Index scan-key row positioning used on table &1	
Message Text:	Index scan-key row positioning used on table &1.
Cause Text:	Index scan-key row positioning is defined as applying selection against the index to position directly to ranges of keys that match some or all of the selection criteria. Index scan-key row positioning only processes a subset of the keys in the index and is a good performing option when selecting a small percentage of rows from the table. The table number refers to the relative position of this table in the query.

SQL4011 - Index scan-key row positioning used on table &1	
Recovery Text:	Refer to "Data access methods" on page 11 for more information about index scan-key row positioning.

SQL4012 - Index created from index &1 for table &2	
Message Text:	Index created from index &1 for table &2.
Cause Text:	A temporary index was created using the specified index to access rows from the queried table for one of the following reasons: -- Perform specified ordering/grouping criteria. -- Perform specified join criteria. The table number refers to the relative position of this table in the query.
Recovery Text:	Creating an index from an index is generally a good performing option. Consider creating a permanent index for frequently run queries. The query can be run in debug mode (STRDBG) to determine the key columns used when creating the index. NOTE: If a permanent index is created, it is possible the query optimizer may still choose to create a temporary index to access the rows from the table.

SQL4013 - Access plan has not been built	
Message Text:	Access plan has not been built.
Cause Text:	An access plan was not created for this query. Possible reasons may include: -- Tables were not found when the program was created. -- The query was complex and required a temporary result table. -- Dynamic SQL was specified.
Recovery Text:	If an access plan was not created, review the possible causes. Attempt to correct the problem if possible.

SQL4014 - &1 join column pair(s) are used for this join position	
Message Text:	&1 join column pair(s) are used for this join position.
Cause Text:	The query optimizer may choose to process join predicates as either join selection or row selection. The join predicates used in join selection are determined by the final join order and the index used. This message indicates how many join column pairs were processed as join selection at this join position. Message SQL4015 provides detail on which columns comprise the join column pairs. If 0 join column pairs were specified then index scan-key row positioning with row selection was used instead of join selection.
Recovery Text:	If fewer join pairs are used at a join position than expected, it is possible no index exists which has keys matching the desired join columns. Try creating an index whose keys match the join predicates. If 0 join column pairs were specified then index scan-key row positioning was used. Index scan-key row positioning is normally a good performing option. Message SQL4011 provides more information on index scan-key row positioning.

SQL4015 - From-column &1.&2, to-column &3.&4, join operator &5, join predicate &6	
Message Text:	From-column &1.&2, to-column &3.&4, join operator &5, join predicate &6.
Cause Text:	Identifies which join predicate was implemented at the current join position. The replacement text parameters are: -- &1: The join 'from table' number. The table number refers to the relative position of this table in the query. -- &2: The join 'from column' name. The column within the join from table which comprises the left half of the join column pair. If the column name is *MAP, the column is an expression (derived field). -- &3: The join 'to table' number. The table number refers to the relative position of this table in the query. -- &4. The join 'to column' name. The column within the join to column which comprises the right half of the join column pair. If the column name is *MAP, the column is an expression (derived field). -- &5. The join operator. Possible values are EQ (equal), NE (not equal), GT (greater than), LT (less than), GE (greater than or equal), LE (less than or equal), and CP (cross join or cartesian product). -- &6. The join predicate number. Identifies the join predicate within this set of join pairs.
Recovery Text:	Refer to "Join optimization" on page 59 for more information about joins.

SQL4016 - Subselects processed as join query	
Message Text:	Subselects processed as join query.
Cause Text:	The query optimizer chose to implement some or all of the subselects with a join query. Implementing subqueries with a join generally improves performance over implementing alternative methods.
Recovery Text:	None

SQL4017 - Host variables implemented as reusable ODP	
Message Text:	Host variables implemented as reusable ODP.
Cause Text:	The query optimizer has built the access plan allowing for the values of the host variables to be supplied when the query is opened. This query can be run with different values being provided for the host variables without requiring the access plan to be rebuilt. This is the normal method of handling host variables in access plans. The open data path (ODP) that will be created from this access plan will be a reusable ODP.
Recovery Text:	Generally, reusable open data paths perform better than non-reusable open data paths.

SQL4018 - Host variables implemented as non-reusable ODP	
Message Text:	Host variables implemented as non-reusable ODP.

SQL4018 - Host variables implemented as non-reusable ODP	
Cause Text:	The query optimizer has implemented the host variables with a non-reusable open data path (ODP).
Recovery Text:	This can be a good performing option in special circumstances, but generally a reusable ODP gives the best performance.

SQL4019 - Host variables implemented as file management row positioning reusable ODP	
Message Text:	Host variables implemented as file management row positioning reusable ODP.
Cause Text:	The query optimizer has implemented the host variables with a reusable open data path (ODP) using file management row positioning.
Recovery Text:	Generally, a reusable ODP performs better than a non-reusable ODP.

SQL402A - Hashing algorithm used to process join	
Message Text:	Hashing algorithm used to process join.
Cause Text:	The hash join algorithm is typically used for longer running join queries. The original query will be subdivided into hash join steps. Each hash join step will be optimized and processed separately. Access plan implementation information for each of the hash join steps is not available because access plans are not saved for the individual hash join dials. Debug messages detailing the implementation of each hash dial can be found in the joblog if the query is run in debug mode using the STRDBG CL command.
Recovery Text:	The hash join method is usually a good implementation choice, however, if you want to disallow the use of this method specify ALWCPYDTA(*YES). Refer to the &qryopt. for more information on hashing algorithm for join processing.

SQL402B - Table &1 used in hash join step &2	
Message Text:	Table &1 used in hash join step &2.
Cause Text:	This message lists the table number used by the hash join steps. The table number refers to the relative position of this table in the query. If there are two or more of these messages for the same hash join step, then that step is a nested loop join. Access plan implementation information for each of the hash join step are not available because access plans are not saved for the individual hash steps. Debug messages detailing the implementation of each hash step can be found in the joblog if the query is run in debug mode using the STRDBG CL command.
Recovery Text:	Refer to "Data access methods" on page 11 for more information about hashing.

SQL402C - Temporary table created for hash join results	
Message Text:	Temporary table created for hash join results.
Cause Text:	The results of the hash join were written to a temporary table so that query processing could be completed. The temporary table was required because the query contained one or more of the following: GROUP BY or summary functions ORDER BY DISTINCT Expression containing columns from more than one table Complex row selection involving columns from more than one table

SQL402C - Temporary table created for hash join results	
Recovery Text:	Refer to "Data access methods" on page 11 for more information about the hashing algorithm for join processing.

SQL402D - Query attributes overridden from query options file &2 in library &1	
Message Text:	Query attributes overridden from query options file &2 in library &1.
Cause Text:	None
Recovery Text:	None

SQL4020 - Estimated query run time is &1 seconds	
Message Text:	Estimated query run time is &1 seconds.
Cause Text:	The total estimated time, in seconds, of executing this query.
Recovery Text:	None

SQL4021 - Access plan last saved on &1 at &2	
Message Text:	Access plan last saved on &1 at &2.
Cause Text:	The date and time reflect the last time the access plan was successfully updated in the program object.
Recovery Text:	None

SQL4022 - Access plan was saved with SRVQRY attributes active	
Message Text:	Access plan was saved with SRVQRY attributes active.
Cause Text:	The access plan that was saved was created while SRVQRY was active. Attributes saved in the access plan may be the result of SRVQRY.
Recovery Text:	The query will be re-optimized the next time it is run so that SRVQRY attributes will not be permanently saved.

SQL4023 - Parallel table prefetch used	
Message Text:	Parallel table prefetch used.
Cause Text:	The query optimizer chose to use a parallel prefetch access method to reduce the processing time required for the table scan.

SQL4023 - Parallel table prefetch used	
Recovery Text:	Parallel prefetch can improve the performance of queries. Even though the access plan was created to use parallel prefetch, the system will actually run the query only if the following are true: -- The query attribute degree was specified with an option of *IO or *ANY for the application process. -- There is enough main storage available to cache the data being retrieved by multiple I/O streams. Normally, 5 megabytes would be a minimum. Increasing the size of the shared pool may improve performance. For more information about parallel table prefetch, refer to "Data access methods" on page 11.

SQL4024 - Parallel index preload access method used

Message Text:	Parallel index preload access method used.
Cause Text:	The query optimizer chose to use a parallel index preload access method to reduce the processing time required for this query. This means that the indexes used by this query will be loaded into active memory when the query is opened.
Recovery Text:	Parallel index preload can improve the performance of queries. Even though the access plan was created to use parallel preload, the system will actually use parallel preload only if the following are true: -- The query attribute degree was specified with an option of *IO or *ANY for the application process. -- There is enough main storage to load all of the index objects used by this query into active memory. Normally, a minimum of 5 megabytes would be a minimum. Increasing the size of the shared pool may improve performance. For more information about parallel table prefetch, refer to "Data access methods" on page 11.

SQL4025 - Parallel table preload access method used	
Message Text:	Parallel table preload access method used.
Cause Text:	The query optimizer chose to use a parallel table preload access method to reduce the processing time required for this query. This means that the data accessed by this query will be loaded into active memory when the query is opened.
Recovery Text:	Parallel table preload can improve the performance of queries. Even though the access plan was created to use parallel preload, the system will actually use parallel preload only if the following are true: -- The query attribute degree must have been specified with an option of *IO or *ANY for the application process. -- There is enough main storage available to load all of the data in the file into active memory. Normally, 5 megabytes would be a minimum. Increasing the size of the shared pool may improve performance. For more information about parallel table prefetch, refer to "Data access methods" on page 11.

SQL4026 - Index only access used on table number &1	
Message Text:	Index only access used on table number &1.
Cause Text:	Index only access is primarily used in conjunction with either index scan-key row positioning index scan-key selection. This access method will extract all of the data from the index rather than performing random I/O to the data space. The table number refers to the relative position of this table in the query.
Recovery Text:	Refer to "Data access methods" on page 11 for more information about index only access.

SQL4027 - Access plan was saved with DB2 Symmetric Multiprocessing installed on the system	
Message Text:	Access plan was saved with DB2 Symmetric Multiprocessing installed on the system.
Cause Text:	The access plan saved was created while the system feature DB2 Symmetric Multiprocessing was installed on the system. The access plan may have been influenced by the presence of this system feature. Having this system feature installed may cause the implementation of the query to change.
Recovery Text:	For more information about how the system feature DB2 Symmetric Multiprocessing can influence a query, refer to the "Controlling parallel processing for queries" on page 194

SQL4028 - The query contains a distributed table	
Message Text:	The query contains a distributed table.
Cause Text:	A distributed table was specified in the query which may cause the query to be processed in multiple steps. If the query is processed in multiple steps, additional messages will detail the implementation for each step. Access plan implementation information for each step is not available because access plans are not saved for the individual steps. Debug messages detailing the implementation of each step can be found in the joblog if the query is run in debug mode using the STRDBG CL command.
Recovery Text:	For more information about how a distributed table can influence the query implementation refer to the Distributed database programming topic collection.

SQL4029 - Hashing algorithm used to process the grouping	
Message Text:	Hashing algorithm used to process the grouping.
Cause Text:	The grouping specified within the query was implemented with a hashing algorithm.
Recovery Text:	Implementing the grouping with the hashing algorithm is generally a performance advantage since an index does not have to be created. However, if you want to disallow the use of this method simply specify ALWCPYDTA(*YES). Refer to "Data access methods" on page 11 for more information about the hashing algorithm.

SQL4030 - &1 tasks specified for parallel scan on table &2	
Message Text:	&1 tasks specified for parallel scan on table &2.
Cause Text:	The query optimizer has calculated the optimal number of tasks for this query based on the query attribute degree. The table number refers to the relative position of this table in the query.

SQL4030 - &1 tasks specified for parallel scan on table &2	
Recovery Text:	Parallel table or index scan can improve the performance of queries. Even though the access plan was created to use the specified number of tasks for the parallel scan, the system may alter that number based on the availability of the pool in which this job is running or the allocation of the table's data across the disk units. Refer to “Data access methods” on page 11 for more information about parallel scan.

SQL4031 - &1 tasks specified for parallel index create over table &2	
Message Text:	&1 tasks specified for parallel index create over table &2.
Cause Text:	The query optimizer has calculated the optimal number of tasks for this query based on the query attribute degree. The table number refers to the relative position of this table in the query.
Recovery Text:	Parallel index create can improve the performance of queries. Even though the access plan was created to use the specified number of tasks for the parallel index build, the system may alter that number based on the availability of the pool in which this job is running or the allocation of the table's data across the disk units. Refer to “Data access methods” on page 11 for more information about parallel index create.

SQL4032 - Index &1 used for bitmap processing of table &2	
Message Text:	Index &1 used for bitmap processing of table &2.
Cause Text:	The index was used, in conjunction with query selection, to create a bitmap. The bitmap, in turn, was used to access rows from the table. This message may appear more than once per table. If this occurs, then a bitmap was created from each index of each message. The bitmaps were then combined into one bitmap using boolean logic and the resulting bitmap was used to access rows from the table. The table number refers to the relative position of this table in the query.
Recovery Text:	The query can be run in debug mode (STRDBG) to determine more specific information. Also, refer to “Data access methods” on page 11 for more information about bitmap processing.

SQL4033 - &1 tasks specified for parallel bitmap create using &2	
Message Text:	&1 tasks specified for parallel bitmap create using &2.
Cause Text:	The query optimizer has calculated the optimal number of tasks to use to create the bitmap based on the query attribute degree.
Recovery Text:	Using parallel index scan to create the bitmap can improve the performance of queries. Even though the access plan was created to use the specified number of tasks, the system may alter that number based on the availability of the pool in which this job is running or the allocation of the file's data across the disk units. Refer to “Data access methods” on page 11 for more information about parallel scan.

SQL4034 - Multiple join classes used to process join	
Message Text:	Multiple join classes used to process join.

SQL4034 - Multiple join classes used to process join	
Cause Text:	Multiple join classes are used when join queries are written that have conflicting operations or cannot be implemented as a single query. Each join class will be optimized and processed as a separate step of the query with the results written out to a temporary table. Access plan implementation information for each of the join classes is not available because access plans are not saved for the individual join class dials. Debug messages detailing the implementation of each join dial can be found in the joblog if the query is run in debug mode using the STRDBG CL command.
Recovery Text:	Refer to "Join optimization" on page 59 for more information about join classes.

SQL4035 - Table &1 used in join class &2	
Message Text:	Table &1 used in join class &2.
Cause Text:	This message lists the table numbers used by each of the join classes. The table number refers to the relative position of this table in the query. All of the tables listed for the same join class will be processed during the same step of the query. The results from all of the join classes will then be joined together to return the final results for the query. Access plan implementation information for each of the join classes are not available because access plans are not saved for the individual classes. Debug messages detailing the implementation of each join class can be found in the joblog if the query is run in debug mode using the STRDBG CL command.
Recovery Text:	Refer to "Join optimization" on page 59 for more information about join classes.

Code license and disclaimer information

Notices

This information was developed for products and services offered in the U.S.A.

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

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

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

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

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

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

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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 Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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

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

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

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

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

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This Database performance and query optimization publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

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

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

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.