DB2

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Performance Guide

DB2 9 BETA

SC10-4222-00

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Performance Guide

DB2 9 BETA

SC10-4222-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

Edition Notice

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Part 1. Introduction to Performance

Chapter 1. Principles of Performance

Elements of performance

Performance is the way a computer system behaves given a particular workload. Performance is measured in terms of system response time, throughput, and availability. Performance is also affected by:

- The resources available in your system
- How well those resources are used and shared.

In general, you tune your system to improve its cost-benefit ratio. Specific goals could include:

- Processing a larger, or more demanding, work load without increasing processing costs

  For example, to increase the work load without buying new hardware or using more processor time
- Obtaining faster system response times, or higher throughput, without increasing processing costs
- Reducing processing costs without degrading service to your users

Translating performance from technical terms to economic terms is difficult. Performance tuning certainly costs money in terms of user time as well as processor time, so before you undertake a tuning project, weigh its costs against its possible benefits. Some of these benefits are tangible:

- More efficient use of resources
- The ability to add more users to the system.

Other benefits, such as greater user satisfaction because of quicker response time, are intangible. All of these benefits should be considered.

Related concepts:

- "Quick-start tips for performance tuning" on page 6
- "Performance tuning guidelines" on page 3

Related tasks:

- "Developing a performance improvement process" on page 5

Performance tuning guidelines

The following guidelines should help you develop an overall approach to performance tuning.

Remember the law of diminishing returns: Your greatest performance benefits usually come from your initial efforts. Further changes generally produce smaller and smaller benefits and require more and more effort.

Do not tune just for the sake of tuning: Tune to relieve identified constraints. If you tune resources that are not the primary cause of performance problems, this has little or no effect on response time until you have relieved the major

constraints, and it can actually make subsequent tuning work more difficult. If there is any significant improvement potential, it lies in improving the performance of the resources that are major factors in the response time.

Consider the whole system: You can never tune one parameter or system in isolation. Before you make any adjustments, consider how it will affect the system as a whole.

Change one parameter at a time: Do not change more than one performance tuning parameter at a time. Even if you are sure that all the changes will be beneficial, you will have no way of evaluating how much each change contributed. You also cannot effectively judge the trade-off you have made by changing more than one parameter at a time. Every time you adjust a parameter to improve one area, you almost always affect at least one other area that you may not have considered. By changing only one at a time, this allows you to have a benchmark to evaluate whether the change does what you want.

Measure and reconfigure by levels: For the same reasons that you should only change one parameter at a time, tune one level of your system at a time. You can use the following list of levels within a system as a guide:

- Hardware
- Operating System
- Application Server and Requester
- Database Manager
- SQL and XQuery Statements
- Application Programs

Check for hardware as well as software problems: Some performance problems may be corrected by applying service either to your hardware, or to your software, or to both. Do not spend excessive time monitoring and tuning your system when simply applying service may make it unnecessary.

Understand the problem before you upgrade your hardware: Even if it seems that additional storage or processor power could immediately improve performance, take the time to understand where your bottlenecks are. You may spend money on additional disk storage only to find that you do not have the processing power or the channels to exploit it.

Put fall-back procedures in place before you start tuning: As noted earlier, some tuning can cause unexpected performance results. If this leads to poorer performance, it should be reversed and alternative tuning tried. If the former setup is saved in such a manner that it can be simply recalled, the backing out of the incorrect information becomes much simpler.

Related concepts:

- "Elements of performance" on page 3
- "Quick-start tips for performance tuning" on page 6

Related tasks:

- "Developing a performance improvement process" on page 5

Developing a performance improvement process

The performance improvement process is an iterative, long term approach to monitoring and tuning aspects of performance. Depending on the result of monitoring, you and your performance team adjust the configuration of the database server and make changes to the applications that use the database server.

Base your performance monitoring and tuning decisions on your knowledge of the kinds of applications that use the data and the patterns of data access. Different kinds of applications have different performance requirements.

Consider the following outline of the performance improvement process as a guideline.

Procedure:

To develop a performance improvement process:

1. Define performance objectives.
2. Establish performance indicators for the major constraints in the system.
3. Develop and execute a performance monitoring plan.
4. Continually analyze the results of monitoring to determine which resources require tuning.
5. Make one adjustment at a time.

   Even if you think that more than one resource requires tuning, or if several tuning options are available for the resource you want to tune, make only one change at a time so that you can make sure that your tuning efforts are producing the effect you want. At some point, you can no longer improve performance by tuning the database server and applications. Then you need to upgrade your hardware.

Actual performance tuning requires trade-offs among system resources. For example, to provide improved I/O performance you might increase buffer pool sizes, but larger buffer pools require more memory, which might degrade other aspects of performance.

Related concepts:

- "Elements of performance" on page 3
- "Performance information that users can provide" on page 6
- "Quick-start tips for performance tuning" on page 6
- "Performance tuning guidelines" on page 3
- "Performance tuning limits" on page 5

Performance tuning limits

Tuning can make only a certain amount of change in the efficiency of a system. Consider how much time and money you should spend on improving system performance, and how much spending additional time and money will help the users of the system.

For example, tuning can often improve performance if the system encounters a performance bottleneck. If you are close to the performance limits of your system and the number of users increases by about ten percent, the response time is likely

to increase by much more than ten percent. In this situation, you need to determine how to counterbalance this degradation in performance by tuning your system.

However, there is a point beyond which tuning cannot help. At this point, consider revising your goals and expectations within the limits of your environment. For significant performance improvements, you might need to add more disk storage, faster CPU, additional CPUs, more main memory, faster communication links, or a combination of these.

Related concepts:

- "Management of database server capacity" in *Administration Guide: Implementation*

Related tasks:

- "Developing a performance improvement process" on page 5

Performance information that users can provide

The first sign that your system requires tuning might be complaints from users. If you do not have enough time to set performance objectives and to monitor and tune in a comprehensive manner, you can address performance by listening to your users. You can usually determine where to start looking for a problem by asking a few simple questions. For example, you might ask your users:

- What do you mean by "slow response"? Is it 10 % slower than you expect it to be, or tens of times slower?
- When did you notice the problem? Is it recent or has it always been there?
- Do other users have the same problem? Are these users one or two individuals or a whole group?
- If a group of users is experiencing the same problems, are they connected to the same local area network?
- Do the the problems seem to be related to a specific transaction or application program?
- Do you notice any pattern in the problem occurrence? For example, does the problem occur at a specific time of day, such as during lunch hour, or is it more or less continuous?

Related concepts:

- "Performance tuning guidelines" on page 3

Related tasks:

- "Developing a performance improvement process" on page 5

Quick-start tips for performance tuning

When you start a new instance of DB2®, consider the following suggestions for a basic configuration:

- Use the Configuration Advisor in the Control Center to get advice about reasonable beginning defaults for your system. The defaults shipped with DB2 should be tuned for your unique hardware environment.

Gather information about the hardware at your site so you can answer the wizard questions. You can apply the suggested configuration parameter settings immediately or let the wizard create a script based on your answers and run the script later.

This script also provides a list of the most commonly tuned parameters for later reference.

- Use other wizards in the Control Center and Client Configuration Assistant for performance-related administration tasks. These tasks are usually those in which you can achieve significant performance improvements by spending spend a little time and effort.

  Other wizards can help you improve performance of individual tables and general data access. These wizards include the Create Database, Create Table, Index, and Configure Multisite Update wizards. The Health Center provides a set of monitoring and tuning tools.
- Use the Design Advisor tool from the Control Center or using the `db2advis` command to find out what indexes, materialized query tables, multi-dimensional clustering tables, and database partitions will improve query performance.
- Use the ACTIVATE DATABASE command to start databases. In a partitioned database, this command activates the database on all database partitions and avoids the startup time required to initialize the database when the first application connects.

  Note: If you use the ACTIVATE DATABASE command, you must shut down the database with the DEACTIVATE DATABASE command. The last application that disconnects from the database does not shut it down.
- Consult the summary tables that list and briefly describe each configuration parameter available for the database manager and each database.

  These summary tables contain a column that indicates whether tuning the parameter results in high, medium,low, or no performance changes, either for better or for worse. Use this table to find the parameters that you might tune for the largest performance improvements.

Related concepts:

- Chapter 15, “The database system monitor information,” on page 353

Related reference:

- “Configuration parameters summary” on page 264

Disk-storage performance factors

The hardware that makes up your system can influence the performance of your system. As an example of the influence of hardware on performance, consider some of the implications associated with disk storage.

Four aspects of disk-storage management affect performance:

- **Division of storage**

  How you divide a limited amount of storage between indexes and data and among table spaces determines to a large degree how each will perform in different situations.
- **Wasted storage**

Wasted storage in itself may not affect the performance of the system that is using it, but wasted storage is a resource that could be used to improve performance elsewhere.

- **Distribution of disk I/O**

  How well you balance the demand for disk I/O across several disk storage devices, and controllers can affect how fast the database manager can retrieve information from disks.

- **Lack of available storage**

  Reaching the limit of available storage can degrade overall performance.

Related concepts:

- "DMS device considerations" in *Administration Guide: Planning*
- "DMS table spaces" in *Administration Guide: Planning*
- "SMS table spaces" in *Administration Guide: Planning*
- "Database directories and files" in *Administration Guide: Planning*
- "Table and index management for MDC tables" on page 337
- "Table and index management for standard tables" on page 333

Part 2. Performance Issues in Database Design

Chapter 2. Planning for Performance in Database Design

Performance enhancing features

Materialized query tables

Materialized query tables (MQTs) are a powerful way to improve response time for complex queries, especially queries that might require some of the following operations:

- Aggregated data over one or more dimensions
- Joins and aggregates data over a group of tables
- Data from a commonly accessed subset of data, that is, from a "hot" horizontal or vertical database partition
- Repartitioned data from a table, or part of a table, in a partitioned database environment

Knowledge of MQTs is integrated into the SQL and XQuery compiler. In the compiler, the query rewrite phase and the optimizer match queries with MQTs and determine whether to substitute an MQT for a query that accesses the base tables. If an MQT is used, the EXPLAIN facility can provide information about which MQT was selected.

Because MQTs behave like regular tables in many ways, the same guidelines for optimizing data access using table space definitions, creating indexes, and issuing RUNSTATS apply to MQTs.

To help you understand the power of MQTs, the following example shows a multidimensional analysis query and how it takes advantage of MQTs.

In this example, assume a database scheme in which a data warehouse contains a set of customers and a set of credit card accounts. The warehouse records the set of transactions that are made with the credit cards. Each transaction contains a set of items that are purchased together. This schema is classified as a multi-star because two large tables, the one containing transaction items and the other identifying the purchase transactions, are together are the hub of the star.

Three hierarchical dimensions that describe a transaction: product, location, and time. The product hierarchy is stored in two normalized tables representing the product group and the product line. The location hierarchy contains city, state, and country or region information and is represented in a single de-normalized table. The time hierarchy contains day, month, and year information and is encoded in a single date field. The date dimensions are extracted from the date field of the transaction using built-in functions. Other tables in this schema represent account information for customers and customer information.

An MQT is created with the sum and count of sales for each level of the following hierarchies:

- Product
- Location
- Time, composed of year, month, day.

Many queries can be satisfied from this stored aggregate data. The following example shows how to create an MQT that computes sum and count of sales along the product group and line dimensions; along the city, state, and country dimension; and along the time dimension. It also includes several other columns in its GROUP BY clause.

```
CREATE TABLE dba.PG_SALESSUM
  AS (
      SELECT l.id AS prodline, pg.id AS pgroup,
             loc.country, loc.state, loc.city,
             l.name AS linename, pg.name AS pgname,
             YEAR(pdate) AS year, MONTH(pdate) AS month,
             t.status,
             SUM(ti.amount) AS amount,
             COUNT(*) AS count
      FROM   cube.transitem AS ti, cube.trans AS t,
             cube.loc AS loc, cube.pgroup AS pg,
             cube.prodline AS l
      WHERE  ti.transid = t.id
         AND ti.pgid = pg.id
         AND pg.lineid = l.id
         AND t.locid = loc.id
         AND YEAR(pdate) > 1990
      GROUP BY l.id, pg.id, loc.country, loc.state, loc.city,
               year(pdate), month(pdate), t.status, l.name, pg.name
     )
DATA INITIALLY DEFERRED REFRESH DEFERRED;

REFRESH TABLE dba.SALESCUBE;
```

Queries that can take advantage of such pre-computed sums would include the following:

- Sales by month and product group
- Total sales for years after 1990
- Sales for 1995 or 1996
- Sum of sales for a product group or product line
- Sum of sales for a specific product group or product line AND for 1995, 1996
- Sum of sales for a specific country.

While the precise answer is not included in the MQT for any of these queries, the cost of computing the answer using the MQT could be significantly less than using a large base table, because a portion of the answer is already computed. MQTs can reduce expensive joins, sorts, and aggregation of base data.

The following sample queries would obtain significant performance improvements because they can use the already computed results in the example MQT.

The first example returns the total sales for 1995 and 1996:

```
SET CURRENT REFRESH AGE=ANY

SELECT YEAR(pdate) AS year, SUM(ti.amount) AS amount
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
   AND YEAR(pdate) IN (1995, 1996)
GROUP BY year(pdate);
```

The second example returns the total sales by product group for 1995 and 1996:

```
SET CURRENT REFRESH AGE=ANY

SELECT pg.id AS "PRODUCT GROUP",
       SUM(ti.amount) AS amount
FROM   cube.transitem AS ti, cube.trans AS t,
       cube.loc AS loc, cube.pgroup AS pg,
       cube.prodline AS l
WHERE  ti.transid = t.id
   AND ti.pgid = pg.id
   AND pg.lineid = l.id
   AND t.locid = loc.id
   AND YEAR(pdate) IN (1995, 1996)
GROUP BY pg.id;
```

The larger the base tables are, the larger the improvements in response time can be because the MQT grows more slowly than the base table. MQTs can effectively eliminate overlapping work among queries by doing the computation once when the MQTs are built and refreshed and reusing their content for many queries.

Related concepts:

- "Replicated materialized query tables in partitioned database environments" on page 175
- "The Design Advisor" on page 43

Table partitioning and multidimensional clustering tables

A table can be both multi-dimensional clustered and partitioned. In a table that is both multi-dimensional clustered and partitioned, columns can be used both in the table partitioning range-partition-spec and in the MDC key. This is useful for achieving a finer granularity of data partition and block elimination than could be achieved by either functionality alone. There are also many applications where it is useful to specify different columns for the MDC key than those on which the table is partitioned. It should be noted that table partitioning is multi-column, while MDC is multi-dimension.

Characteristics of a mainstream DB2 V9.1 data warehouse:

The following recommendations were focused on typical, mainstream warehouses that are new for DB2 V9.1. The following characteristics are assumed:

- The database runs on multiple machines or multiple AIX logical partitions.
- Database partitioning feature (DPF) is used (tables are created using the DISTRIBUTE BY HASH clause).
- There are four to fifty data partitions.
- The table for which MDC and table partitioning is being considered is a major fact table.
- The table has 100 million to 100 billion rows.
- New data is loaded at various time frames: nightly, weekly, monthly.
- Daily ingest volume is 10 thousand to 10 million records.
- Data volumes vary: The biggest month is 5X the size of the smallest month. Likewise, the biggest dimensions (product line, region) have a 5X size range.
- 1 to 5 years of detailed data is retained.
- Expired data is rolled out monthly or quarterly.
- Tables use a wide range of query types. However, the workload is mostly analytical queries with the following characteristics, relative to OLTP workloads:

  - larger results sets with up to 2 million rows
  - most or all queries are hitting views, not base tables
- SQL clauses selecting data by ranges (BETWEEN clause), items in lists, and so on.

Characteristics of a mainstream DB2 V9.1 data warehouse fact table:

A typical warehouse fact table, might use the following design:

- Create data partitions on the Month column.
- Define a data partition for each period you roll-out, for example, 1 month, 3 months.
- Create MDC dimensions on Day and on 1 to 4 additional dimensions. Typical dimensions are: product line and region.
- All data partitions and MDC clusters are spread across all database partitions.

MDC and table partitioning provide overlapping sets of benefits. The following table lists potential needs in your organization and identifies a recommended organization scheme based on the characteristics identified previously.

Table 1. Using table partitioning with MDC tables

Issue	Recommended scheme	Recommendation
Data availability during roll-out	Table partitioning	Use the DETACH PARTITION clause to roll out large amounts of data with minimal disruption.
Query performance	Table partitioning and MDC	MDC is best for querying multiple dimensions. Table partitioning helps through data partition elimination.
Minimal reorganization	MDC	MDC maintains clustering, which reduces the need to reorganize.
Rollout a month or more of data during a traditional offline window	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data during a micro-offline window (less than 1 minute)	Table partitioning	Data partitioning addresses this need fully. MDC adds nothing and would be less suitable on its own.
Rollout a month or more of data while keeping the table fully available for business users submitting queries without any loss of service.	MDC	MDC only addresses this need somewhat. Table partitioning would not be suitable due to the short period the table goes offline.
Load data daily (either on-line or offline)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.
Load data ″continually″ (on-line)	Table partitioning and MDC	MDC provides most of the benefit here. Table partitioning provides incremental benefits.

Table 1. Using table partitioning with MDC tables (continued)

Issue	Recommended scheme	Recommendation
Query execution performance for ″traditional BI″ queries	Table partitioning and MDC	MDC is especially good for querying cubes/multiple dimensions. Table partitioning helps via partition elimination.
Minimize reorganization pain, by avoiding the need for reorganization or reducing the pain associated with performing the task	MDC	MDC maintains clustering which reduces the need to reorg. If MDC is used, data partitioning does not provide incremental benefits. However if MDC is not used, range partitioning helps reduce the need for reorg by maintaining some course grain clustering at the partition level.

Example 1:

Consider a table with key columns YearAndMonth and Province. A reasonable approach to planning this table might be to partition by date with 2 months per data partition. In addition, you might also organize by Province, so that all rows for a particular province within any two month date range are clustered together, as shown in Figure 1 on page 16.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);
```

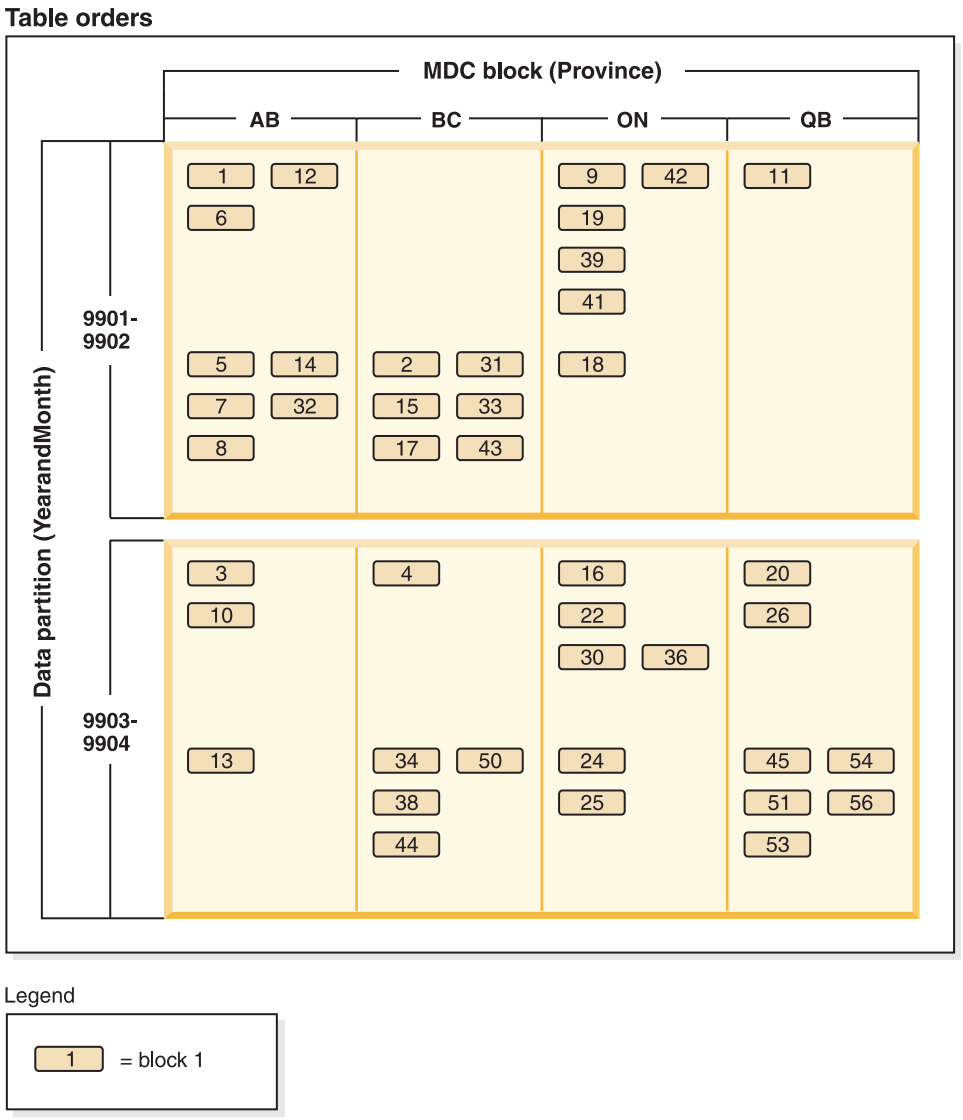


Figure 1. A table partitioned by YearAndMonth and organized by Province

Example 2:

Finer granularity can be achieved by adding YearAndMonth to the ORGANIZE BY clause, as shown in Figure 2 on page 17.

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (YearAndMonth, Province);
```

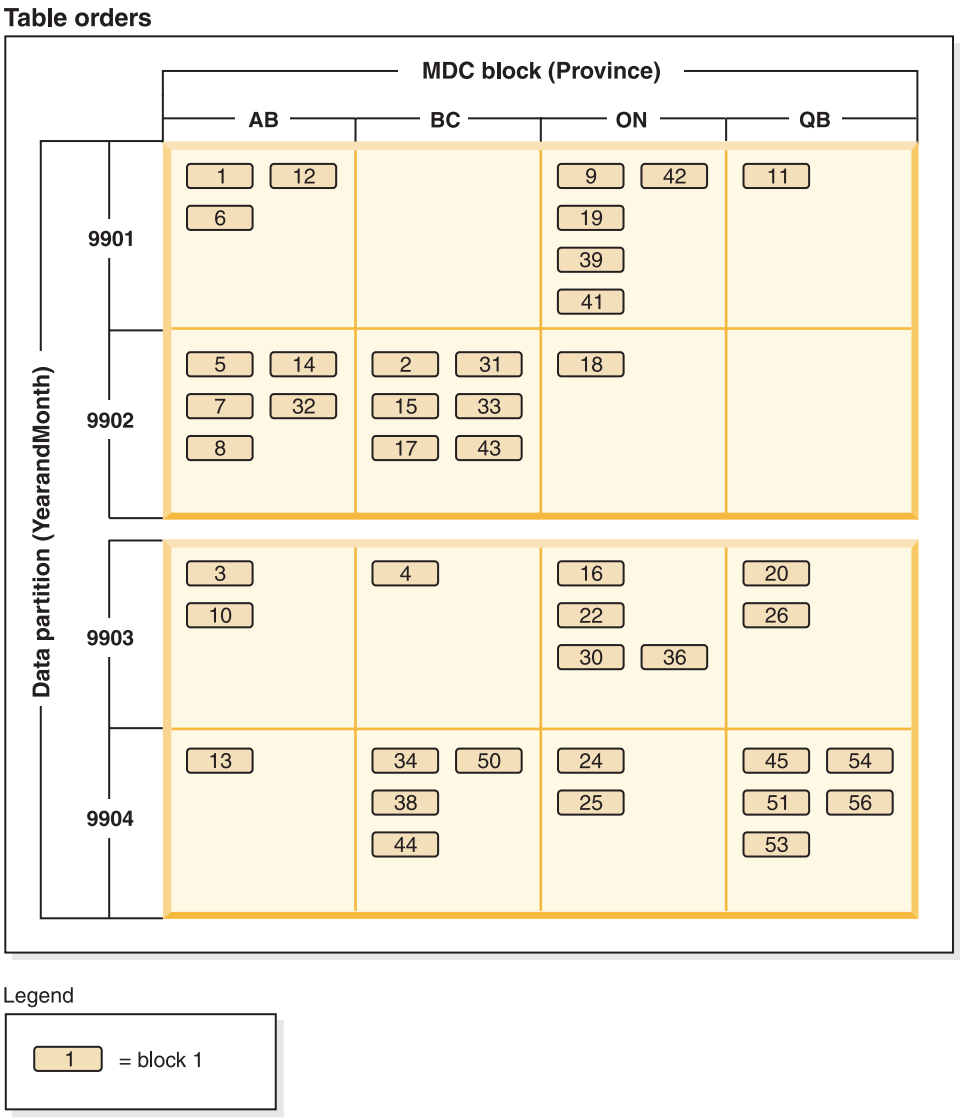


Figure 2. A table partitioned by YearAndMonth and organized by Province and YearAndMonth

In cases where the partitioning is such that there is only a single value in each range, nothing is gained by including the table partitioning column in the MDC key.

Considerations:

- Compared to a basic table, both MDC tables and partitioned tables require more storage. These storage needs are additive but are considered reasonable given the benefits.
- If you choose not to combine table partitioning and MDC functionality in your partitioned database environment, table partitioning is best in cases where you can confidently predict the data distribution, which is generally the case for the types of systems discussed here. Otherwise, MDC should be considered.

Related concepts:

- "Data partitions" in *Administration Guide: Planning*
- "Designing multidimensional clustering (MDC) tables" in *Administration Guide: Planning*

- "Multidimensional clustering (MDC) table creation, placement, and use" in *Administration Guide: Planning*
- "Optimization strategies for MDC tables" on page 18
- "Optimization strategies for partitioned tables" on page 19
- "Partitioned materialized query table behavior" in *Administration Guide: Implementation*
- "Partitioned tables" in *Administration Guide: Planning*
- "Table and index management for MDC tables" on page 337
- "Table partitioning" in *Administration Guide: Planning*

Related tasks:

- "Altering partitioned tables" in *Administration Guide: Implementation*
- "Approaches to defining ranges on partitioned tables" in *Administration Guide: Implementation*
- "Approaches to migrating existing tables and views to partitioned tables" in *Administration Guide: Implementation*
- "Creating partitioned tables" in *Administration Guide: Implementation*

Related reference:

- "Restrictions on native XML data store" in *XML Guide*

Optimization strategies for MDC tables

If you create multi-dimensional clustering (MDC) tables, the performance of many queries might improve because the optimizer can apply additional optimization strategies. These strategies are primarily based on the improved efficiency of block indexes, but the advantage of clustering on more than one dimension also permits faster data retrieval.

Note: MDC table optimization strategies can also implement the performance advantages of intra-partition parallelism and inter-partition parallelism.

Consider the following specific advantages of MDC tables:

- Dimension block index lookups can identify the required portions of the table and quickly scan only the required blocks.
- Because block indexes are smaller than RID indexes, lookups are faster.
- Index ANDing and ORing can be performed at the block level and combined with RIDs.
- Data is guaranteed to be clustered on extents, which makes retrieval faster.
- Rows can be deleted faster when roll out can be used.

When conditions are met to allow roll out, a more efficient way to delete rows from MDC tables is used. The conditions are:

- The DELETE statement is searched, not positioned (that is, does not use the "WHERE CURRENT OF" clause).
- No WHERE clause (all rows are to be deleted) or the only conditions in the WHERE clause are on dimensions.
- There are no tables with DATA CAPTURE CHANGES.
- The table is not the parent in referential integrity, does not have on delete triggers defined, and is not used in any MQTs that are refreshed immediately.

- A cascaded delete operation may qualify for roll out, if its foreign key is a subset of its dimension columns.
- The DELETE statement cannot appear in a SELECT statement in an OLD TABLE clause.

For a roll out deletion, the deleted records are not logged. Instead, the pages that contain the records are made to look empty by reformatting parts of the pages. The changes to the reformatted parts are logged, but the records themselves are not logged. There is no change in the logging of index updates, so the performance improvement depends on how many RID indexes there are. The fewer RID indexes, the better the improvement is, as a percentage of the total time and log space.

An estimate of the amount of space saved in the log can be made with this formula,

```
S + 38*N - 50*P
```

where `N` is the number of records deleted, `S` is total size of the records deleted, including overhead such as null indicators and varchar lengths, and `P` is the number of pages in the blocks containing the records deleted:

This figure is the amount of space saved in the permanent log, and needs to be doubled to include the savings in reserved space in the active log.

To disable roll out behavior in deletions, set the DB2_MDC_ROLLOUT registry variable to ″OFF″.

Consider the following simple example for an MDC table named **sales** with dimensions defined on the **region** and **month** columns:

```
SELECT * FROM SALES
    WHERE MONTH='March' AND REGION='SE'
```

For this query, the optimizer can perform a dimension block index lookup to find blocks in which the month of March and the SE region occur. Then it can quickly scan only the resulting blocks of the table to fetch the result set.

Related concepts:

- “Table and index management for MDC tables” on page 337

Optimization strategies for partitioned tables

Data partition elimination refers to the database servers ability to determine, based on the query predicates, that only a subset of the data partitions of a table need to be accessed to answer a query. Data partition elimination offers particular benefit when running decision support queries against a partitioned table.

A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. Data from a given table is partitioned into multiple storage objects based on the specifications provided in the PARTITION BY clause of the CREATE TABLE statement. These storage objects can be in different table spaces, in the same table space, or a combination of both.

The following example demonstrates the performance benefits of data partition elimination. If you issue the following statement:

```
CREATE TABLE custlist(subsdate DATE, Province CHAR(2), AccountID INT)
PARTITION BY RANGE(subsdate)
(STARTING FROM '1/1/1990' IN ts1,
STARTING FROM '1/1/1991' IN ts1,
STARTING FROM '1/1/1992' IN ts1,
STARTING FROM '1/1/1993' IN ts2,
STARTING FROM '1/1/1994' IN ts2,
STARTING FROM '1/1/1995' IN ts2,
STARTING FROM '1/1/1996' IN ts3,
STARTING FROM '1/1/1997' IN ts3,
STARTING FROM '1/1/1998' IN ts3,
STARTING FROM '1/1/1999' IN ts4,
STARTING FROM '1/1/2000' IN ts4,
STARTING FROM '1/1/2001' ENDING '12/31/2001' IN ts4);
```

Assume you are interested in customer information for the year 2000. If you issue the following query:

```
SELECT * FROM custlist WHERE subsdate BETWEEN '1/1/2000' AND '12/31/2000';
```

As Figure 3 shows, the database server determines that only one data partition in table space 4 (ts4) must be accessed to resolve this query.

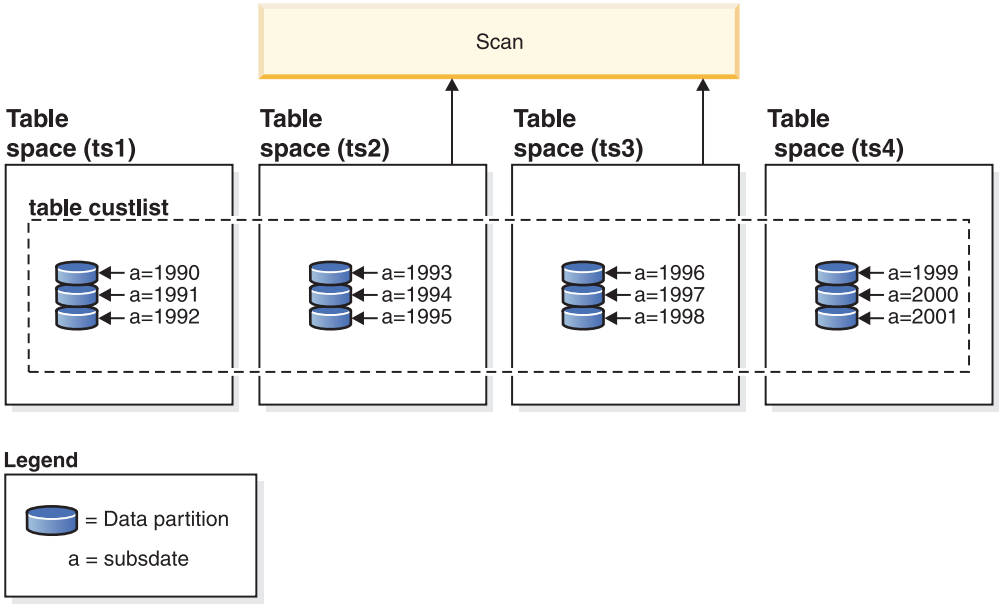


Figure 3. The performance benefits of data partition elimination on a partitioned table

Another example of data partition elimination shown in figure Figure 4 on page 21, is an index scan involving two indexes and based on the following scheme:

```
CREATE TABLE multi (sale_date date, region char(2))
PARTITION BY (sale_date)
(STARTING '01/01/2005' ENDING '12/31/2005' EVERY 1 MONTH);
CREATE INDEX sx ON multi(sale_date);
CREATE INDEX rx ON multi(region);
```

If you issue the following query:

```
SELECT * FROM multi WHERE
sale_date BETWEEN '6/1/2005' AND '7/31/2005' AND REGION = 'NW';
```

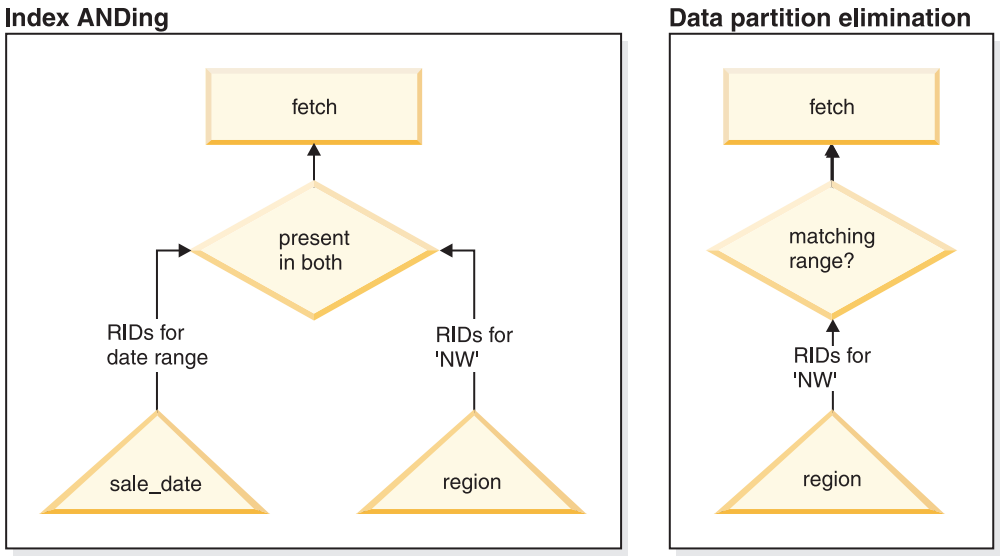


Figure 4. Optimizer decision path for both table partitioning and index ANDing

Without table partitioning, one likely plan is index ANDing. Index ANDing performs the following tasks:

- Reads all relevant index entries from each index
- Saves both sets of row identifiers (RIDs)
- Matches RIDs to determine which occur in both indexes
- Uses the RIDs to fetch the rows

As Figure 4 demonstrates, with table partitioning, the index is read to find matches for both region and sale_date, allowing for fast retrieval of matching rows.

DB2 Explain:

You can also use DB2 Explain to determine the partition elimination chosen by the DB2 optimizer. The **DP Elim Predicates** information shows which data partitions are scanned to resolve the following query:

```
SELECT * FROM custlist WHERE subsdate
BETWEEN '12/31/1999' AND '1/1/2001'

Arguments:
---------
DPESTFLG: (Number of data partitions accessed are Estimated)
    FALSE
DPLSTPRT: (List of data partitions accessed)
    9-11
DPNUMPRT: (Number of data partitions accessed)
   3

DP Elim Predicates:
------------------
Range 1)
  Stop  Predicate: (Q1.A <= '01/01/2001')
 Start Predicate: ('12/31/1999' <= Q1.A)



Objects Used in Access Plan:
 --------------------------

  Schema: MRSRINI
```

```
 Name:    CUSTLIST
 Type:    Data Partitioned Table
  Time of creation:      2005-11-30-14.21.33.857039
  Last statistics update:    2005-11-30-14.21.34.339392
   Number of columns:      3
   Number of rows:      100000
  Width of rows:      19
  Number of buffer pool pages:    1200
   Number of data partitions:    12
   Distinct row values:      No
   Tablespace name:      <VARIOUS>
```

Multi-column support:

Data partition elimination works for cases where multiple columns are used as the table partitioning key.

For example, if you issue the following statement:

```
CREATE TABLE sales(year INT, month INT)
 PARTITION BY RANGE(year, month)
 (STARTING FROM (2001, 1) ENDING AT(2001,3) IN ts1,
 ENDING AT(2001,6) IN ts2,
 ENDING AT(2001,9) IN ts3,
 ENDING AT(2001,12) IN ts4,
 ENDING AT(2002,3) IN ts5,
 ENDING AT(2002,6) IN ts6,
 ENDING AT(2002,9) IN ts7,
 ENDING AT(2002,12) IN ts8)
```

Next, issue the following query:

```
 SELECT * FROM sales WHERE year = 2001 AND month < 8
```

The query optimizer deduces that only data partitions in ts1, ts2 and ts3 must be accessed to resolve this query.

Note: In the case where multiple columns make up the table partitioning key, data partition elimination is only possible when you have predicates on the leading columns of the composite key, since the non-leading columns used for the table partitioning key are not independent.

Multi-range support:

It is possible to achieve data partition elimination on data partitions with multiple ranges (that is, OR'ed together). Using the table created in the previous example, execute the following query:

```
 SELECT * FROM sales
 WHERE (year = 2001 AND month <= 3) OR (year = 2002 and month >= 10)
```

The database server only accesses data for the first quarter of 2001 and the last quarter of 2002.

Generated columns:

You can use generated columns as table partitioning keys.

For example, you can issue the following statement:

```
CREATE TABLE sales(a INT, b INT GENERATED ALWAYS AS (a / 5))
 IN ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9,ts10
 PARTITION BY RANGE(b)
 (STARTING FROM (0) ENDING AT(1000) EVERY (50))
```

In this case, predicates on the generated column are used for data partition elimination. In addition, when the expression used to generate the columns is monotonic, the database server translates predicates on the source columns into predicates on the generated columns, which enables data partition elimination on the generated columns.

For example, if you have the following query:

```
SELECT * FROM sales WHERE a > 35
```

The database server generates an extra predicate on b (b > 7) from a (a > 35), thus allowing data partition elimination.

Join predicates:

Join predicates can also be used in data partition elimination, if the join predicate is pushed down to the table access level. The join predicate is only pushed down to the table access level on the inner of a nested loop join (NLJN).″

Consider the following tables:

```
CREATE TABLE T1(A INT, B INT)
 PARTITION BY RANGE(A, B)
 (STARTING FROM (1, 1)
 ENDING (1,10) IN ts1, ENDING (1,20) IN ts2,
 ENDING (2,10) IN ts3, ENDING (2,20) IN ts4,
  ENDING (3,10) IN ts5, ENDING (3,20) IN ts6,
 ENDING (4,10) IN ts7, ENDING (4,20) IN ts8)

CREATE TABLE T2 (A INT, B INT)
```

Predicates used:

```
P1: T1.A = T2.A
P2: T1.B > 15
```

In this example, the exact data partitions that will be accessed at compile time cannot be determined, due to unknown outer values of the join. In this case, as well as cases where host variables or parameter markers are used, data partition elimination occurs at runtime when the necessary values are bound.

During runtime when T1 is the inner of a NLJN, data partition elimination occurs dynamically, based on the predicates, for every outer value of T2.A. During runtime the predicates T1.A = 3 and T1.B > 15 are applied for the outer value T2.A = 3, which qualifies the data partitions in table spaces ts6 and ts7 to be accessed.

Consider that column A in tables T1 and T2 have the following values:

Outer table T2: column A	Inner table T1: column A	Inner table T1: column B	Inner table T1: data partition location
2	3	20	ts6
3	2	10	ts3
3	2	18	ts4
	3	15	ts6
	1	40	ts3

To perform a nested-loop join (assuming a table scan for the inner table), the database manager performs the following steps:

1. Reads the first row from T2. The value for A is 2.
2. Binds T2.A value (which is 2) to the column T2.A in the join predicate T1.A = T2.A. The predicate becomes T1.A = 2.
3. Applies data partition elimination using the predicates T1.A = 2 and T1.B > 15. This qualifies data partitions in table spaces ts4 and ts5.
4. Scans the data partitions in table spaces ts4 and ts5 of table T1 until a row is found after applying T1.A = 2 and T1.B > 15. The first row found that qualifies is row 3 of T1.
5. Joins the matching row.
6. Scans the data partitions in table spaces ts4 and ts5 of table T1 until the next match (T1.A = 2 and T1.B > 15) is found. No more rows are found.
7. Repeats steps 1 through 6 for next row (replacing the value of A with 3) of T2 until all the rows from T2 are exhausted.

Related concepts:

- "Block and row identifier preparation information" on page 645
- "Configuration parameters that affect query optimization" on page 158
- "Data organization schemes in DB2 and Informix databases" in *Administration Guide: Planning*
- "Explain information for data objects" on page 227
- "Explain tools" on page 222
- "Guidelines for analyzing explain information" on page 233
- "Guidelines for capturing explain information" on page 231
- "Guidelines for using explain information" on page 223
- "Miscellaneous explain information" on page 650
- "Partitioned tables" in *Administration Guide: Planning*
- "Explain facility" on page 221
- "Table partitioning" in *Administration Guide: Planning*

Related reference:

- "Explain tables" on page 585
- "CREATE TABLE statement" in *SQL Reference, Volume 2*

Chapter 3. Indexes

Using relational indexes to improve performance

Indexes can be used to improve performance when accessing table data. Relational indexes are used when working with relational data. For XML data access, indexes over XML data are used.

Although the optimizer decides whether to use a relational index to access relational table data, except in the following case, you must decide which indexes might improve performance and create these indexes. Exceptions are the dimension block indexes and the composite block index that are created automatically for each dimension that you specify when you create a multi-dimensional clustering (MDC) table.

You must also execute the RUNSTATS utility to collect new statistics about the relational indexes in the following circumstances:

- After you create a relational index
- After you change the prefetch size

You should also execute the RUNSTATS utility at regular intervals to keep the statistics current. Without up-to-date statistics about indexes, the optimizer cannot determine the best data-access plan for queries.

Note: To determine whether a relational index is used in a specific package, use the Explain facility. To plan relational indexes, use the Design Advisor from the Control Center or the `db2advis` tool to get advice about relational indexes that might be used by one or more SQL statements.

Advantages of a relational index over no index

If no index exists on a table, a table scan must be performed for each table referenced in an SQL query. The larger the table, the longer a table scan takes because a table scan requires each table row to be accessed sequentially. Although a table scan might be more efficient for a complex query that requires most of the rows in a table, an index scan can access table rows more efficiently for a query that returns only some table rows.

The optimizer chooses an index scan if the relational index columns are referenced in the SELECT statement and if the optimizer estimates that an index scan will be faster than a table scan. Index files generally are smaller and require less time to read than an entire table, particularly as tables grow larger. In addition, the entire index may not need to be scanned. The predicates that are applied to the index reduce the number of rows to be read from the data pages.

If an ordering requirement on the output can be matched with an index column, then scanning the index in column order will allow the rows to be retrieved in the correct order without a sort.

Each relational index entry contains a search-key value and a pointer to the row containing that value. If you specify the ALLOW REVERSE SCANS parameter in the CREATE INDEX statement, the values can be searched in both ascending and

descending order. It is therefore possible to bracket the search, given the right predicate. A relational index can also be used to obtain rows in an ordered sequence, eliminating the need for the database manager to sort the rows after they are read from the table.

In addition to the search-key value and row pointer, a relational index can contain include columns, which are non-indexed columns in the indexed row. Such columns might make it possible for the optimizer to get required information only from the index, without accessing the table itself.

Note: The existence of a relational index on the table being queried does not guarantee an ordered result set. Only an ORDER BY clause ensures the order of a result set.

Although indexes can reduce access time significantly, they can also have adverse effects on performance. Before you create indexes, consider the effects of multiple indexes on disk space and processing time:

- Each index requires storage or disk space. The exact amount depends on the size of the table and the size and number of columns in the relational index.
- Each INSERT or DELETE operation performed on a table requires additional updating of each index on that table. This is also true for each UPDATE operation that changes the value of an index key.
- The LOAD utility rebuilds or appends to any existing relational indexes.

  The `indexfreespace` MODIFIED BY parameter can be specified on the LOAD command to override the index PCTFREE used when the relational index was created.
- Each relational index potentially adds an alternative access path for an SQL query for the optimizer to consider, which increases the compilation time.

Choose indexes carefully to address the needs of the application program.

Related concepts:

- "Relational index planning tips" on page 28
- "Index reorganization" on page 318
- "Relational index performance tips" on page 30
- "Table and index management for MDC tables" on page 337
- "Table and index management for standard tables" on page 333
- "Table reorganization" on page 317
- "The Design Advisor" on page 43
- "Index cleanup and maintenance" on page 340
- "Space requirements for indexes" in *Administration Guide: Planning*

Related tasks:

- "Collecting catalog statistics" on page 279
- "Collecting index statistics" on page 282
- "Creating an index" in *Administration Guide: Implementation*

Index structure

The database manager uses a B+ tree structure for index storage. A B+ tree has one or more levels, as shown in the following diagram, in which `RID` means row ID:

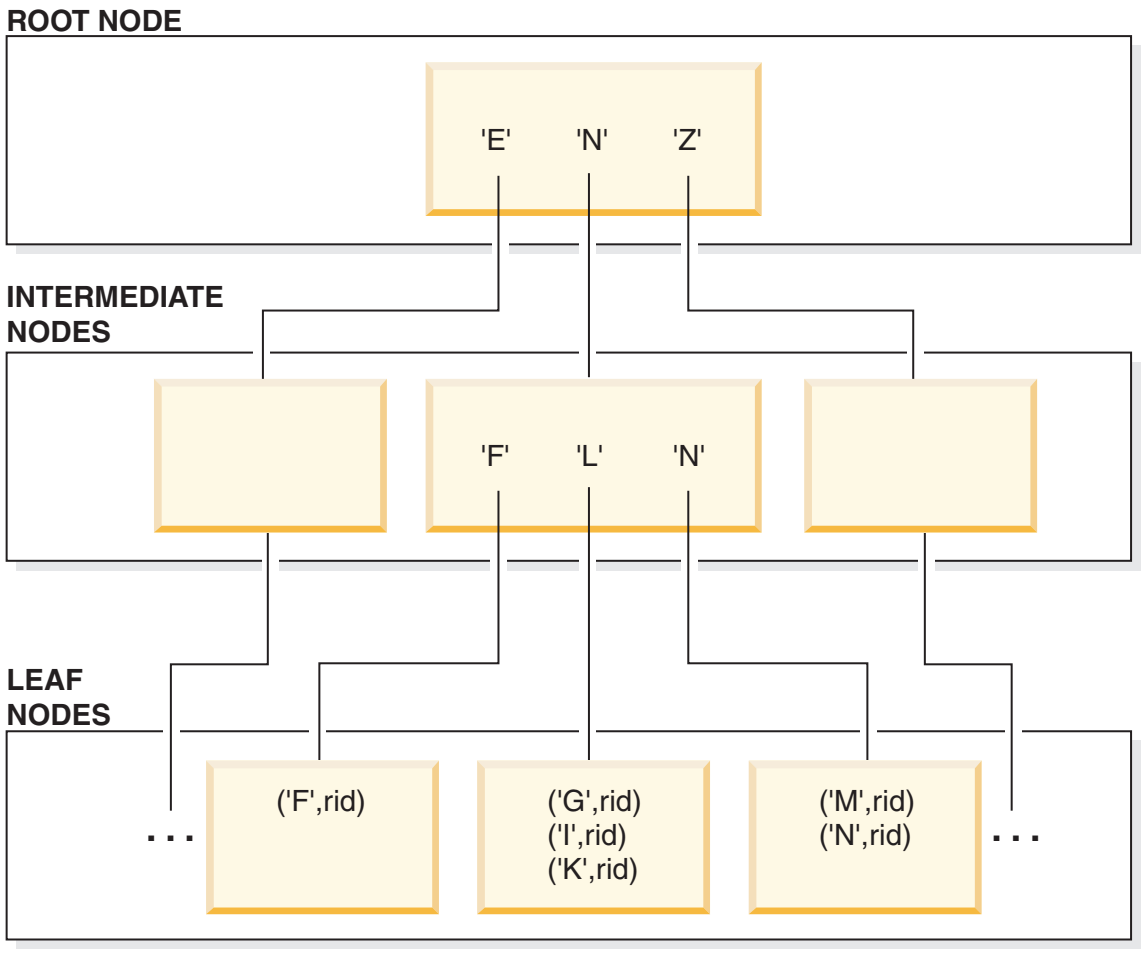


Figure 5. B+ Tree Structure

The top level is called the *root node*. The bottom level consists of *leaf nodes* in which the index key values are stored with pointers to the row in the table that contains the key value. Levels between the root and leaf node levels are called *intermediate nodes*.

When it looks for a particular index key value, the index manager searches the index tree, starting at the root node. The root contains one key for each node at the next level. The value of each of these keys is the largest existing key value for the corresponding node at the next level. For example, if an index has three levels as shown in the figure, then to find an index key value, the index manager searches the root node for the first key value greater than or equal to the key being looked for. The root node key points to a specific intermediate node. The index manager follows this procedure through the intermediate nodes until it finds the leaf node that contains the index key that it needs.

The figure shows the key being looked for as "I". The first key in the root node greater than or equal to "I" is "N". This points to the middle node at the next level. The first key in that intermediate node that is greater than or equal to "I" is "L". This points to a specific leaf node where the index key for "I" and its corresponding row ID is found. The row ID identifies the corresponding row in the base table. The leaf node level can also contain pointers to previous leaf nodes. These pointers allow the index manager to scan across leaf nodes in either

direction to retrieve a range of values after it finds one value in the range. The ability to scan in either direction is only possible if the index was created with the ALLOW REVERSE SCANS clause.

For multi-dimensional clustering (MDC) tables, a block index is created automatically for each clustering dimension that you specify for the table. An additional composite block index is also created, which contains a key part for each column involved in any dimension of the table. These indexes contain pointers to block IDs (BIDs) instead of RIDs, and provide data-access improvements.

In DB2 Version 8.1 and later, indexes can be of either type 1 or type 2. A *type-1 index* is the older index style. Indexes that you created in earlier versions of DB2 are of this kind.

A type-2 index is somewhat larger than a type-1 index and provides features that minimize next-key locking. The one-byte *ridFlag* byte stored for each RID on the leaf page of a type-2 index is used to mark the RID as logically deleted so that it can be physically removed later. For each variable length column included in the index, one additional byte stores the actual length of the column value. Type-2 indexes might also be larger than type-1 indexes because some keys might be marked deleted but not yet physically removed from the index page. After the DELETE or UPDATE transaction is committed, the keys marked deleted can be cleaned up.

Related concepts:

- "Using relational indexes to improve performance" on page 25
- "Index reorganization" on page 318
- "Online index defragmentation" on page 341

Relational index planning tips

The relational indexes that you create should depend on the relational data and the queries that access it.

Use the Design Advisor from the Control Center or the `db2advis` tool to find the best indexes for a specific query or for the set of queries that defines a workload. This tool recommends relational indexes with such performance enhancing features as INCLUDE columns, inherited unique indexes, and ALLOW REVERSE SCANS indexes.

The following guidelines can help you determine how to create useful relational indexes for various purposes:

- To avoid some sorts, define primary keys and unique keys, wherever possible, by using the CREATE UNIQUE INDEX statement.
- To improve data-retrieval, add INCLUDE columns to unique indexes. Good candidates are columns that:
  - Are accessed frequently and therefore would benefit from index-only access
  - Are not required to limit the range of index scans
  - Do not affect the ordering or uniqueness of the index key.
- To access small tables efficiently, use relational indexes to optimize frequent queries to tables with more than a few data pages, as recorded in the NPAGES column in the SYSCAT.TABLES catalog view. You should:
  - Create an index on any column you will use when joining tables.

  - Create an index on any column from which you will be searching for particular values on a regular basis.
- To search efficiently, decide between ascending and descending ordering of keys depending on the order that will be used most often. Although the values can be searched in reverse direction if you specify the ALLOW REVERSE SCANS parameter in the CREATE INDEX statement, scans in the specified index order perform slightly better than reverse scans.
- To save index maintenance costs and space:
  - Avoid creating relational indexes that are partial keys of other index keys on the columns. For example, if there is an index on columns a, b, and c, then a second index on columns a and b is not generally useful.
  - Do not create relational indexes arbitrarily on all columns. Unnecessary indexes not only use space, but also cause large prepare times. This is especially important for complex queries, when an optimization class with dynamic programming join enumeration is used.

    Use the following general rule for the typical number of relational indexes that you define for a table. This number is based on the primary use of your database:
    - For online transaction processing (OLTP) environments, create one or two indexes
    - For read-only query environments, you might create more than five indexes
    - For mixed query and OLTP environments, you might create between two and five indexes.
- To improve performance of delete and update operations on the parent table, create relational indexes on foreign keys.
- To improve performance of DELETE and UPDATE operations involving IMMEDIATE and INCREMENTAL MQTs, create unique relational indexes on the implied unique key of the MQT, which is the columns in the GROUP BY clause of the MQT definition.
- For fast sort operations, create relational indexes on columns that are frequently used to sort the relational data.
- To improve join performance with a multiple-column relational index, if you have more than one choice for the first key column, use the column most often specified with the "=" (equijoin) predicate or the column with the greatest number of distinct values as the first key.
- To help keep newly inserted rows clustered according to an index and avoid page splits, define a clustering index. A clustering index should significantly reduce the need for reorganizing the table.

  Use the PCTFREE keyword when you define the table to specify how much free space should be left on the page to allow inserts to be placed appropriately on pages. You can also specify the `pagefreespace` MODIFIED BY clause of the LOAD command.
- To enable online index defragmentation, use the MINPCTUSED option when you create relational indexes. MINPCTUSED specifies the threshold for the minimum amount of used space on an index leaf page as well as enabling online index defragmentation. This might reduce the need for reorganization at the cost of a performance penalty during key deletions if these deletions physically remove keys from the index page.

Consider creating a relational index in the following circumstances:

- Create a relational index on columns that are used in WHERE clauses of the queries and transactions that are most frequently processed.

  The WHERE clause:

  ```
  WHERE WORKDEPT='A01' OR WORKDEPT='E21'
  ```

will generally benefit from an index on WORKDEPT, unless the WORKDEPT column contains many duplicate values.

- Create a relational index on a column or columns to order the rows in the sequence required by the query. Ordering is required not only in the ORDER BY clause, but also by other features, such as the DISTINCT and GROUP BY clauses.

  The following example uses the DISTINCT clause:

  ```
  SELECT DISTINCT WORKDEPT
    FROM EMPLOYEE
  ```

  The database manager can use an index defined for ascending or descending order on WORKDEPT to eliminate duplicate values. This same index could also be used to group values in the following example with a GROUP BY clause:

  ```
  SELECT WORKDEPT, AVERAGE(SALARY)
    FROM EMPLOYEE
  GROUP BY WORKDEPT
  ```

- Create a relational index with a compound key that names each column referenced in a statement. When an index is specified in this way, relational data can be retrieved from the index only, which is more efficient than accessing the table.

  For example, consider the following SQL statement:

  ```
  SELECT LASTNAME
    FROM EMPLOYEE
    WHERE WORKDEPT IN ('A00','D11','D21')
  ```

  If a relational index is defined for the WORKDEPT and LASTNAME columns of the EMPLOYEE table, the statement might be processed more efficiently by scanning the index than by scanning the entire table. Note that since the predicate is on WORKDEPT, this column should be the first column of the relational index.

- Create a relational index with INCLUDE columns to improve the use of indexes on tables. Using the previous example, you could define a unique relational index as:

  ```
  CREATE UNIQUE INDEX x ON employee (workdept) INCLUDE (lastname)
  ```

  Specifying lastname as an INCLUDE column rather than as part of the index key means that lastname is stored only on the leaf pages of the index.

Related concepts:

- "Multidimensional clustering (MDC) table creation, placement, and use" in *Administration Guide: Planning*
- "Using relational indexes to improve performance" on page 25
- "Index reorganization" on page 318
- "Online index defragmentation" on page 341
- "Relational index performance tips" on page 30
- "The Design Advisor" on page 43

Relational index performance tips

Consider the following suggestions for using and managing relational indexes:

- **Specify parallelism at index creation and reorganization**

  When you create or reorganize relational indexes on large tables hosted by an SMP machine, consider setting *intra_parallel* to YES (1) or SYSTEM (-1) to take advantage of parallel performance improvements.

Multiple processors can be used to scan and sort relational data.

- **Specify a large utility heap**

  Write access by other users or applications to the underlying table is supported for both CREATE INDEX and REORG INDEXES. When you expect a lot of update activity on the underlying table for the relational index being created or reorganized, consider configuring a large utility heap. A large utility heap will speed up the index creation or index reorganization during the catch up phase. All writing activity on the index or indexes being created or reorganized is logged in the DB2 logs and in the internal memory buffer space. The internal memory buffer space is a designated memory area allocated on demand from the utility heap to store the changes to the index being created or reorganized. It is the use of this memory that allows the catch up phase to work quickly. The allocated memory is freed once the create or reorganization operations complete. Ensuring that there is enough utility heap to accommodate all or most of the changes to the indexes that are being created or reorganized can have a very positive performance impact on the catch up phase.

- **Specify separate table spaces for relational indexes**

  Indexes can be stored in a different table space from the table data. This can allow for more efficient use of disk storage by reducing the movement of read/write heads during index access. You can also create index table spaces on faster physical devices. In addition, you can assign the index table space to a different buffer pool, which might keep the index pages in the buffer longer because they do not compete with table data pages.

  When you do not place indexes in separate table spaces, both data and index pages use the same extent size and prefetch quantity. If you use a different table space for indexes, you can select different values for all the characteristics of a table space. Because indexes are usually smaller than tables and are spread over fewer containers, indexes often have smaller extent sizes, such as 8 and 16 pages. The query optimizer considers the speed of the device for a table space when it chooses an access plan.

- **Ensure the degree of clustering**

  If your SQL statement requires ordering (for example, if it contains ORDER BY, GROUP BY, and DISTINCT SQL clauses), the optimizer might not choose the index even though it satisfies the ordering in the following cases:
  - Index clustering is poor. For information, examine the CLUSTERRATIO and CLUSTERFACTOR columns of SYSCAT.INDEXES.
  - The table is so small that it is cheaper to scan the table and sort the answer set in memory.
  - There are competing indexes for accessing the table.

  After you create a clustering index, perform a REORG TABLE in classic mode, which creates a perfectly organized index. To recluster the table, you might perform a sort and LOAD instead. In general, a table can only be clustered on one index. Build additional indexes after you build the clustering index. A clustering index attempts to maintain a particular order of data, improving the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility.

  To help maintain the clustering ratio, specify an appropriate PCTFREE when you alter a table before you load or reorganize that table. The free space on each page specified by PCTFREE provides space for inserts, so that these inserts can be clustered appropriately. If you do not specify PCTFREE for the table, reorganization eliminates all extra space.

  Note: Clustering is not currently maintained during updates unless you are using range-clustered tables. That is, if you update a record so that its key

value changes in the clustering index, the record is not necessarily moved to a new page to maintain the clustering order. To maintain clustering, use DELETE and then INSERT instead of UPDATE.

- **Keep table and index statistics up-to-date**

  After you create a new relational index, run the RUNSTATS utility to collect index statistics. These statistics allow the optimizer to determine whether using the index can improve access performance.

- **Enable online index defragmentation**

  Online index defragmentation is enabled if the MINPCTUSED clause is set to greater than zero for the relational index. Online index defragmentation allows indexes to be compacted by merging leaf pages when the free space on a page falls at or below the specified level while the index remains available.

- **Reorganize relational indexes as necessary**

  To get the best performance from your indexes, consider reorganizing your indexes periodically because updates to tables can cause index page prefetch to become less effective.

  To reorganize the index, either drop it and re-create it or use the REORG utility.

  To reduce the need for frequent reorganization, when you create a relational index specify an appropriate PCTFREE to leave a percentage of free space on each index leaf page as it is created. During future activity, records can be inserted into the index with less likelihood of causing index page splits. Page splits cause index pages not to be contiguous or sequential, which in turn results in decreased efficiency of index page prefetching.

  Note: The PCTFREE specified when you create the relational index is retained when the index is reorganized.

  Dropping and re-creating or reorganizing the relational index also creates a new set of pages that are roughly contiguous and sequential and improves index page prefetch. Although more costly in time and resources, the REORG TABLE utility also ensures clustering of the data pages. Clustering has greater benefit for index scans that access a significant number of data pages.

  In a symmetric multi-processor (SMP) environment, if the *intra_parallel* database manager configuration parameter is YES or ANY, the "classic" REORG TABLE mode, which uses a shadow table for fast table reorganization, can use multiple processors to rebuild the indexes.

- **Analyze EXPLAIN information about relational index usage**

  Periodically, run EXPLAIN on your most frequently used queries and verify that each of your relational indexes is used at least once. If an index is not used in any query, consider dropping that index.

  EXPLAIN information also lets you see if table scans on large tables are processed as the inner table of nested loop joins. If they are, an index on the join-predicate column is either missing or considered ineffective for applying the join predicate.

- **Use volatile tables for tables that vary widely in size**

  A *volatile* table is a table that might vary in size at run time from empty to very large. For this kind of table, in which the cardinality varies greatly, the optimizer might generate an access plan that favors a table scan instead of an index scan.

  Declaring a table "volatile" using the ALTER TABLE...VOLATILE statement allows the optimizer to use an index scan on the volatile table. The optimizer will use an index scan instead of a table scan regardless of the statistics in the following circumstances:

  – All columns referenced are in the index

– The index can apply a predicate in the index scan.

If the table is a typed table, using the ALTER TABLE...VOLATILE statement is supported only on the root table of the typed table hierarchy.

Related concepts:

- "Index access and cluster ratios" on page 169
- "Using relational indexes to improve performance" on page 25
- "Relational index planning tips" on page 28
- "Index reorganization" on page 318
- "Index structure" on page 26
- "Online index defragmentation" on page 341
- "Table and index management for MDC tables" on page 337
- "Table and index management for standard tables" on page 333
- "Table reorganization" on page 317
- "Index cleanup and maintenance" on page 340

Related reference:

- "intra_parallel - Enable intra-partition parallelism " on page 491

XML data indexing overview

An index over XML data can be used to improve the efficiency of queries on XML documents that are stored in an XML column.

In contrast to traditional relational indexes, where index keys are composed of one or more table columns you specify, an index over XML data uses a particular XML pattern expression to index paths and values in XML documents stored within a single column. The data type of that column must be XML.

Instead of providing access to the beginning of a document, index entries in an index over XML data provide access to nodes within the document by creating index keys based on XML pattern expressions. Because multiple parts of a XML document can satisfy an XML pattern, multiple index keys may be inserted into the index for a single document.

You create an index over XML data using the CREATE INDEX statement, and drop an index over XML data using the DROP INDEX statement. The GENERATE KEY USING XMLPATTERN clause you include with the CREATE INDEX statement specifies what you want to index.

Some of the keywords used with the CREATE INDEX statement for indexes on non-XML columns do not apply to indexes over XML data. The UNIQUE keyword also has a different meaning for indexes over XML data.

Example: Creating an index over XML data: Suppose that table companyinfo has an XML column named companydocs, which contains XML document fragments like these:

Document for Company1:

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
```

```
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Document for Company2:

```
<company name="Company2">
  <emp id="31664" salary="60000" gender="Male">
    <name>
      <first>Chris</first>
      <last>Murphy</last>
    </name>
    <dept id="M55">
      Marketing
    </dept>
  </emp>
  <emp id="42366" salary="50000" gender="Female">
    <name>
      <first>Nicole</first>
      <last>Murphy</last>
    </name>
    <dept id="K55">
      Sales
    </dept>
  </emp>
</company>
```

Users of the companyinfo table often retrieve employee information using the employee ID. You might use an index like this one to make that retrieval more efficient:

```
CREATE INDEX empindex on companyinfo(companydocs)        1
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'       2
   AS SQL DOUBLE                                    3
```

Figure 6. Example of an index over XML data

Notes to Figure 6:

1 The index over XML data is defined on the companydocs column of the companyinfo table. companydocs must be of the XML data type.

2 The GENERATE KEY USING XMLPATTERN clause provides information about what you want to index. This clause is called an XML index specification. The XML index specification contains an XML pattern clause. The XML pattern clause in this example indicates that you want to index the values of the id attribute of each employee element.

3 AS SQL DOUBLE indicates that indexed values are stored as DOUBLE values.

Related reference:

- "CREATE INDEX statement" in *SQL Reference, Volume 2*

Indexes in partitioned tables

Understanding index behavior on partitioned tables

Indexes on partitioned tables are similar to indexes for ordinary tables, that is each index contains pointers to rows in all the data partitions of the table. One

important difference however is that each index on a partitioned table is an independent object. In partitioned database environments, the index is distributed across the database partitions in the same manner as the table. Because an index on a partitioned table can act independently of other indexes, some special considerations are needed with respect to which table space is used when creating an index on a partitioned table.

An index on a partitioned table is created in a single table space even if the table's data partitions span multiple table spaces. Both DMS and SMS table spaces support the use of indexes in a different location than the table. All table spaces specified must be in the same database partition group. Each index can be placed in its own table space, including large table spaces. Each index table space must use the same storage mechanism as the data partitions, either DMS or SMS. Indexes in large table spaces can contain up to 2^29 pages.

Additional benefits of an index on a partitioned table include:

- Improved performance of drop index and online index create.
- The ability to use different values for any of the table space characteristics between each index on the table (for example, different page sizes for each index may be appropriate to ensure better space utilization).
- Reduced IO contention providing more efficient concurrent access to the index data for the table.
- When individual indexes are dropped space will immediately become available to the system without the need for an index reorganization.
- If you choose to perform index reorganization, an individual index can be reorganized.

Figure 7 shows a non-partitioned index on a partitioned table residing in a single table space.

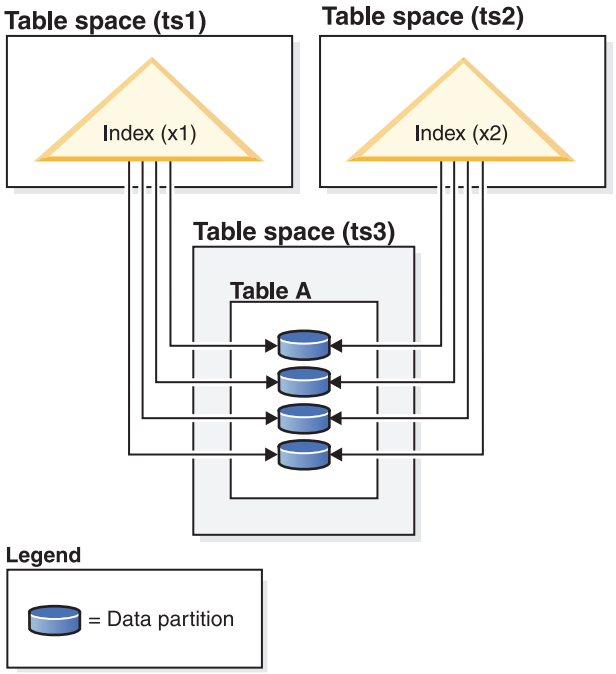


Figure 7. Index behavior on a partitioned table

Figure 8 shows index behavior on a partitioned table that is also distributed across multiple database partitions.

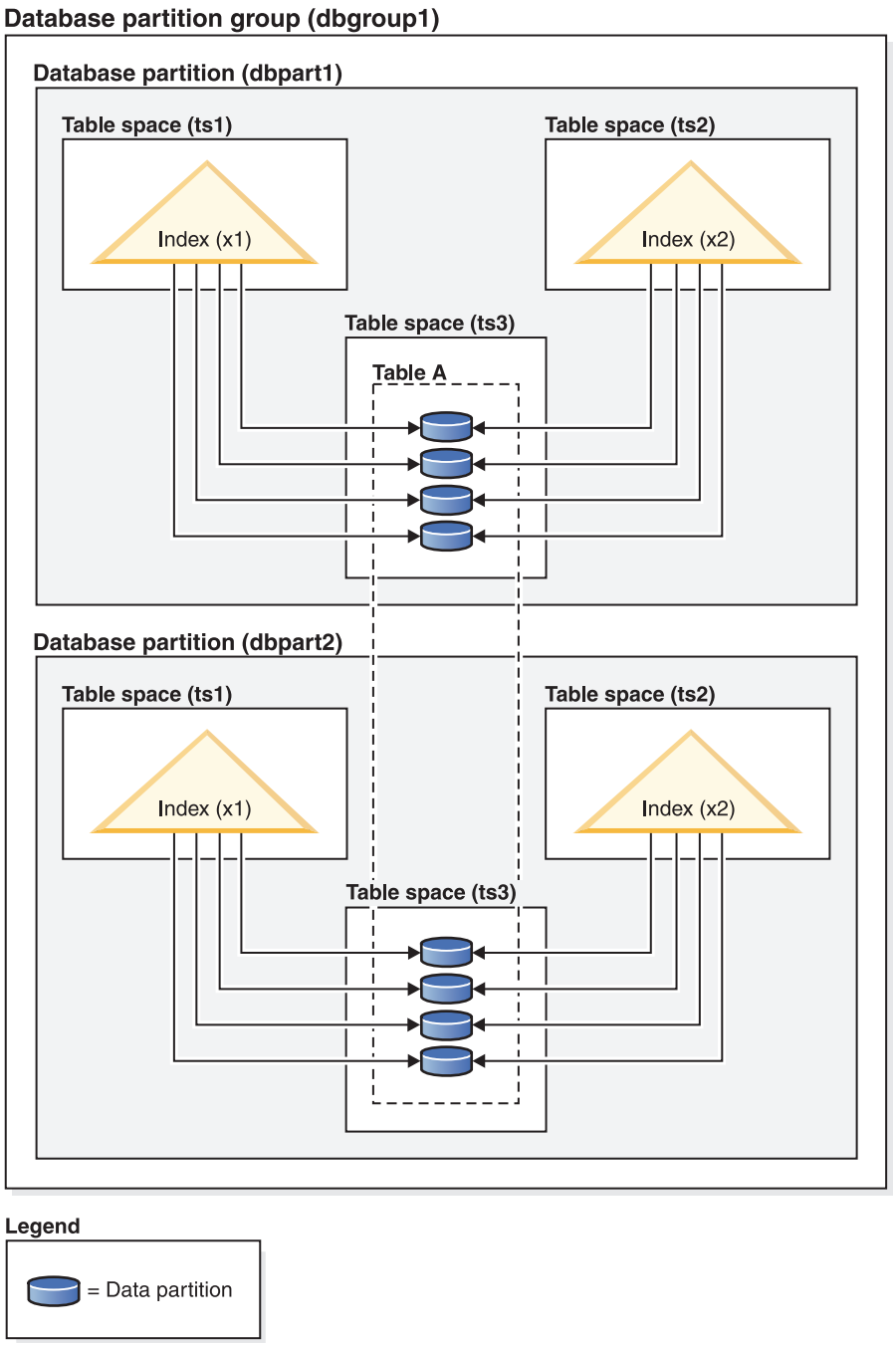


Figure 8. Index behaviour on a table that is both distributed and partitioned.

You can specify an index table space for a partitioned table in the CREATE INDEX ...IN <tbspace1> statement, which can be different from the index table space specified in the CREATE TABLE .. INDEX IN <tbspace2> statement.

For partitioned tables only, you can override the index location by using the IN clause on the CREATE INDEX statement, which allows you to specify a table space location for the index. This approach allows you to place different indexes on a partitioned table in a different table space as required. When you create a

partitioned table without specifying where to place its non-partitioned indexes, and you create an index using the CREATE INDEX statement which does not specify a specific table space, the index is created in the table space of the first attached or visible data partition. Each of the following three possible cases is evaluated in order, starting with case 1, to determine where the index is created. The evaluation stops when there is a match to one of the cases:

```
Case 1:
When an index table space is specified in CREATE INDEX ... IN
<tbspace1> statement the table space specified in <tbspace1>
is used for this index.
Case 2:
When an index table space is specified in the CREATE TABLE .. INDEX IN
<tbspace2> statement the table space specified in <tbspace2>
is used for this index.
Case 3:
When no table space is specified, use the table space used by the first
attached or visible data partition.
```

Where the index is created depends on what is done during the CREATE TABLE statement. For non-partitioned tables, if you do not specify any INDEX IN clause, the database fills it in for you and is the same as your data table space. For partitioned tables, if you leave it blank, it remains as blank, and case 3 applies.

Example 1: This example assumes the existence of a partitioned table sales (a int, b int, c int), and creates a unique index 'a_idx' in the table space 'ts1'.

```
CREATE UNIQUE INDEX a_idx ON sales ( a ) IN ts1
```

Example 2: This example assumes the existence of a partitioned table sales (a int, b int, c int), and creates an index 'b_idx' in the table space 'ts2'.

```
CREATE INDEX b_idx ONsales ( b ) IN ts2
```

Related concepts:

- "Using relational indexes to improve performance" on page 25
- "Understanding clustering index behavior on partitioned tables" on page 38
- "Indexes" in *SQL Reference, Volume 1*
- "Large object behavior in partitioned tables" in *SQL Reference, Volume 1*
- "Options on the CREATE INDEX statement" in *Administration Guide: Implementation*
- "Partitioned tables" in *Administration Guide: Planning*
- "Space requirements for indexes" in *Administration Guide: Planning*

Related tasks:

- "Altering partitioned tables" in *Administration Guide: Implementation*
- "Creating partitioned tables" in *Administration Guide: Implementation*
- "Enabling parallelism when creating indexes" in *Administration Guide: Implementation*

Related reference:

- "CREATE INDEX statement" in *SQL Reference, Volume 2*
- "REORG INDEXES/TABLE command" in *Command Reference*
- "Command Line Processor (CLP) samples" in *Samples Topics*

Understanding clustering index behavior on partitioned tables

Clustering indexes offer the same benefits for partitioned tables as they do for regular tables. However, care must be taken in choosing a clustering index with regards to the table partitioning key definitions.

You can create clustering indexes on a partitioned table using any clustering key. The database server attempts to use the clustering index to cluster data locally within each data partition. During a clustered insert, a lookup is done in the index to look for a suitable row identifier (RID). This RID is used as a starting point when looking for space in the table to insert the record. To achieve well-maintained clustering with good performance, there should be a correlation between the index columns and the table partitioning key columns. One way to ensure such correlation is to prefix the index columns by the table partitioning key columns, as shown in the following example :

```
PARTITION BY RANGE (Month, Region)
CREATE INDEX ...(Month, Region, Department) CLUSTER
```

Although the database server does not enforce this correlation, there is an expectation that all keys in the index are grouped together by partition IDs to achieve good clustering. For example, if a table is partitioned on quarter, and a clustering index is defined on date, because the relation between quarter and date exists, optimal clustering of the data with good performance can be achieved because all keys of any data partition are grouped together within the index.

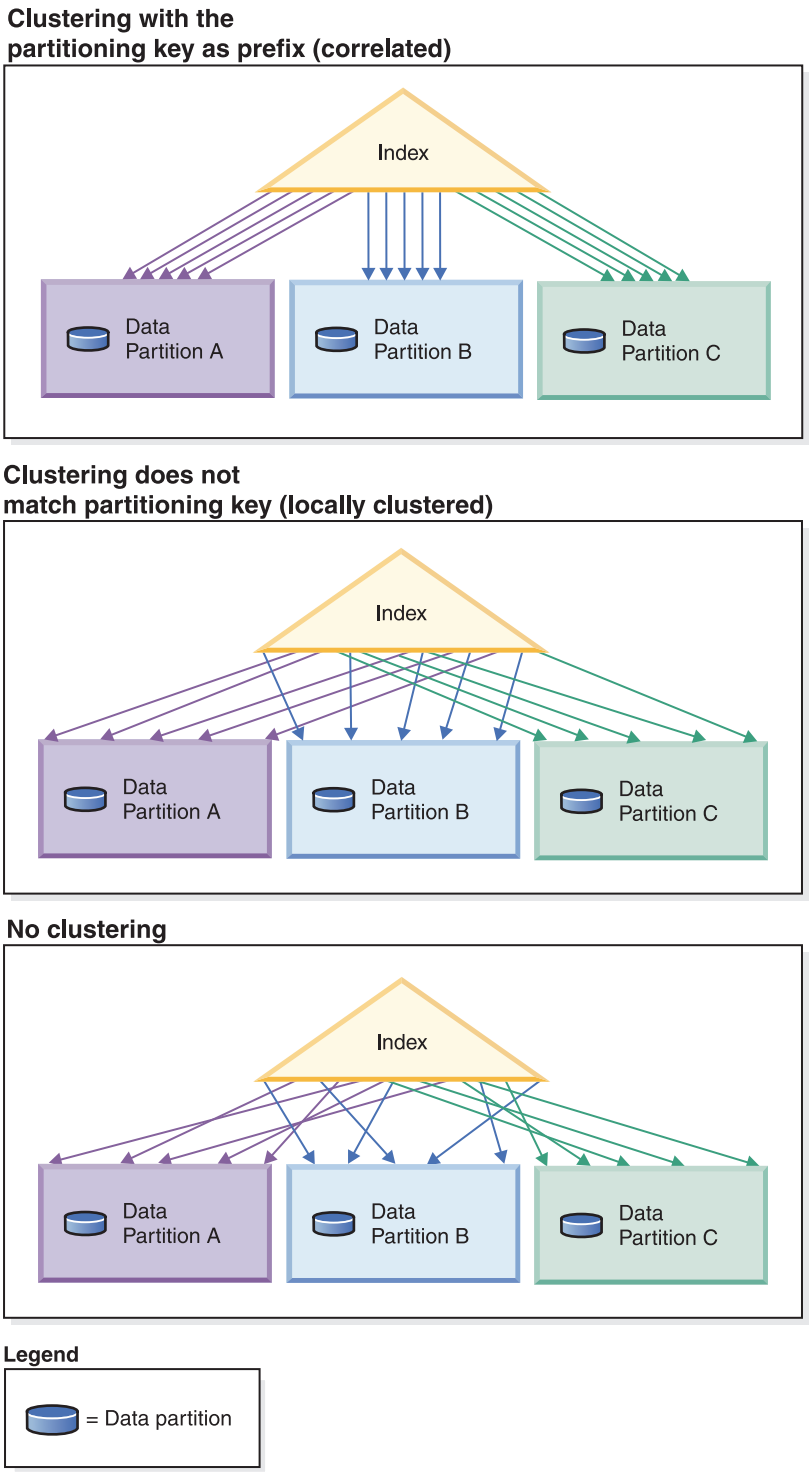


Figure 9. The possible effects of a clustered index on a partitioned table. In the first figure, data is both globally and locally clustered.

As Figure 9 shows, given the layout of the index and data in each example, optimal scan performance is achieved when the clustering correlates to the table partitioning key. When clustering is not correlated to the table partitioning key it is unlikely that the index will be locally clustered. Because a correlation between the table partitioning columns and index columns is expected, a perfect locally clustered scenerio is highly unlikely.

Benefits of clustering include:

- Within each data partition, rows are in key order.
- Clustering indexes improve the performance of scans that traverse the table in the order of the keys. This is because the scan fetches the first row of the first page, then each row in that same page until it has fetched all of the rows for that page and moves on to the next. This means that only one page of the table needs to be in the buffer pool at any given time. In contrast, if the table is not clustered, then each row fetched is likely to be from a different page. Unless there is room in the buffer pool to hold the entire table, this will result in each page being fetched more than once, greatly slowing the scan.

  For partitioned tables, the ideal case of fetching each page only once during the scan can be guaranteed only if the table partitioning key is a prefix of the clustering key (see first figure in Figure 9 on page 39). However, if the clustering key is not correlated to the table partitioning key as described previously, and the data is locally clustered, you can still achieve the full benefit of the clustered index if there is enough space to hold one page of each data partition in the buffer pool. This is because each row fetched for a given data partition is near the previous row fetched for that same data partition. (see the second figure of Figure 9 on page 39). As previously mentioned, the clustering may not be well maintained in the case where the clustering key is unrelated to the table partitioning key, but if you do not expect a high level of insert, update and delete activity on your table this approach should be beneficial.

  Even if there is not sufficient space for a page of every data partition to be held in the buffer pool, there is still some benefit to be gained from defining a clustered index.

Related concepts:

- "Using relational indexes to improve performance" on page 25
- "Optimization strategies for partitioned tables" on page 19
- "Partitioned tables" in *Administration Guide: Planning*
- "Clustered indexes" in *SQL Guide*
- "Non-clustered indexes" in *SQL Guide*

Related tasks:

- "Altering partitioned tables" in *Administration Guide: Implementation*
- "Creating partitioned tables" in *Administration Guide: Implementation*
- "Creating an index" in *Administration Guide: Implementation*
- "Creating indexes" in *SQL Guide*
- "Designing clustered indexes" in *SQL Guide*
- "Designing non-clustered indexes" in *SQL Guide*

Related reference:

- "CREATE INDEX statement" in *SQL Reference, Volume 2*

Asynchronous index cleanup

Asynchronous index cleanup (AIC) is the deferred cleanup of indexes following operations that invalidate index entries. Depending on the type of index, the entries may by row identifiers (RIDs) or block identifiers (BIDs). Either way, these entries are removed by the index cleaners which operate asynchronously in the background.

AIC accelerates the detach of a data partition from a partitioned table. If the partitioned table contains one or more non-partitioned indexes then AIC is initiated. In this case, AIC removes all non-partitioned index entries that refer to the detached data partition and any pseudo-deleted entries. Once all the indexes have been cleaned, the identifier associated with the detached data partition is removed from the system catalog.

Note: If the partitioned table has dependent materialized query tables (MQTs) defined, AIC is not initiated until after a SET INTEGRITY operation is performed.

While AIC is in progress, normal table access is maintained. Queries accessing the indexes simply ignore any invalid entries that have not yet been cleaned.

In most cases, one cleaner is started for each non-partitioned index associated with the partitioned table. An internal task distribution daemon is responsible for distributing the AIC tasks to the appropriate database partitions and assigning database agents.

Both the distribution daemon and cleaner agents are internal, system applications. They appear in the LIST APPLICATION output with the application name **db2taskd** and **db2aic**, respectively. To prevent accidental disruption, system applications cannot be forced. The distribution daemon remains online as long as the database is active. The cleaners remain activate until the cleaning is complete. If the database deactivates while cleaning is in progress, AIC resumes when the database reactivates.

Performance:

AIC incurs minimal performance impact.

An instantaneous row lock test is required to determine whether a pseudo-deleted entry is committed. However, since the lock is never acquired, concurrency is not affected.

Each cleaner acquires a minimal table space lock (IX) and table lock (IS), the locks are released when the cleaner determines other applications are waiting for the locks. When this occurs, the cleaner temporarily suspends processing for five minutes.

Cleaners are also integrated with the utility throttling facility. By default, each cleaner has a utility impact priority of 50. This priority can be changed using the SET UTIL_IMPACT_PRIORITY command or the db2UtilityControl API.

Monitoring:

While AIC is in progress, it can be monitored with the LIST UTILITIES command. Each index cleaner appears in the monitor as a separate utility.

Example 1

The following example demonstrates AIC activity in the WSDB database at the current database partition using the Command Line Processor (CLP) interface.

```
$ db2 list utilities show detail

ID                                = 2
```

```
Type                            = ASYNCHRONOUS INDEX CLEANUP
Database Name                   = WSDB
Partition Number                = 0
Description                     = Table: USER1.SALES, Index: USER1.I2
Start Time                      = 12/15/2005 11:15:01.967939
State                           = Executing
Invocation Type                 = Automatic
Throttling:
   Priority                     = 50
Progress Monitoring:
      Total Work                = 5 pages
      Completed Work            = 0 pages
      Start Time                = 12/15/2005 11:15:01.979033

ID                              = 1
Type                            = ASYNCHRONOUS INDEX CLEANUP
Database Name                   = WSDB
Partition Number                = 0
Description                     = Table: USER1.SALES, Index: USER1.I1
Start Time                      = 12/15/2005 11:15:01.978554
State                           = Executing
Invocation Type                 = Automatic
Throttling:
   Priority                     = 50
Progress Monitoring:
      Total Work                = 5 pages
      Completed Work            = 0 pages
      Start Time                = 12/15/2005 11:15:01.980524
```

In this case, there are two cleaners operating on the USERS1.SALES table. One cleaner is processing index I1, the other, index I2.

The progress monitoring section shows the estimated total number of index pages that need cleaning and the current number of clean index pages.

The State field indicates the cleaner's current state. Normally, the state is Executing. It may be in Waiting state if the cleaner is waiting to be assigned to an available database agent, or if the cleaner is temporarily suspended due to lock contention.

Note: Different tasks on different database partitions can have the same utility ID, since each database partition assigns IDs to tasks on that database partition only.

Related concepts:

- "Attributes of detached data partitions" in *Administration Guide: Implementation*
- "Data partitions" in *Administration Guide: Planning*

Related tasks:

- "Detaching a data partition" in *Administration Guide: Implementation*

Related reference:

- "Guidelines and restrictions on altering partitioned tables with attached or detached data partitions" in *Administration Guide: Implementation*
- "ALTER TABLE statement" in *SQL Reference, Volume 2*
- "SET UTIL_IMPACT_PRIORITY command" in *Command Reference*

Chapter 4. Design Advisor

The Design Advisor

The DB2 Design Advisor is a tool that can help you significantly improve your workload performance. The task of selecting which indexes, MQTs, clustering dimensions, or database partitions to create for a complex workload can be quite daunting. The Design Advisor identifies all of the objects needed to improve the performance of your workload. Given a set of SQL statements in a workload, the Design Advisor will generate recommendations for:

- New indexes
- New materialized query tables (MQTs)
- Conversion to multidimensional clustering (MDC) tables
- Redistribution of tables
- Deletion of indexes and MQTs unused by the specified workload (through the GUI tool)

You can have the Design Advisor implement some or all of these recommendations immediately or schedule them for a later time.

Using either the Design Advisor GUI or the command-line tool, the Design Advisor can help simplify the following tasks:

Planning for or setting up a new database

While designing your database use the Design Advisor to:

- Generate design alternatives in a test environment of a partitioned database environment, and of indexes, MQTs, and MDC tables.
- For partitioned database environments, you can use the Design Advisor to:
  - Determine the database partitioning strategy before loading data into a database.
  - Assist in migrating from a single-partition DB2 database to a multiple-partition DB2 database.
  - Assist in migrating from another database product to a multiple-partition DB2 database.
- Evaluate indexes, MQTs, MDC tables, or database partitioning strategies that have been generated manually.

Workload performance tuning

After your database is set up, you can use the Design Advisor to:

- Improve performance of a particular statement or workload.
- Improve general database performance, using the performance of a sample workload as a gauge.
- Improve performance of the most frequently executed queries, for example, as identified by the Activity Monitor.
- Determine how to optimize the performance of a new key query.
- Respond to Health Center recommendations regarding shared memory utility or sort heap problems in a sort-intensive workload.
- Find objects that are not used in a workload.

Design Advisor output

If you use the Design Advisor GUI, you can view, save, or implement the recommendations from within the Design Advisor. If you run the Design Advisor from the command line, the output is printed to stdout by default, and saved in the ADVISE_TABLE and ADVISE_INDEX tables:

- The ADVISE_TABLE contains USE_TABLE='Y' for MQT, MDC tables, and database partitioning strategy recommendations.

  The MQT recommendations can also be found in the ADVISE_MQT table; the MDC recommendations can also be found in the ADVISE_TABLE table; and the database partitioning strategy recommendations can also be found in the ADVISE_PARTITION table. The RUN_ID value in these tables corresponds to the START_TIME value in the ADVISE_INSTANCE table for each execution of the Design Advisor.
- The ADVISE_INDEX table contains USE_INDEX='Y' or 'R' for index recommendations.

The ADVISE_INSTANCE table is also updated with one row each time that the Design Advisor runs:

- The START_TIME field and the END_TIME field show the start and stop times of the utility, respectively.
- The STATUS field will contain 'COMPLETED' if the utility ended successfully.
- The MODE field indicates whether the -m option was used.
- The COMPRESSION field indicates the type of compression used.

You can save the Design Advisor recommendations to a file using the -o option. The saved Design Advisor output consists of the following elements:

- CREATE STATEMENTS given for new indexes, MQTs, database partitioning strategies, and MDC tables.
- REFRESH statements for MQTs.
- RUNSTATS commands for new objects.
- Existing MQTs and indexes will appear in the recommended script if they were and are used to execute the workload.

Note: The COLSTATS column of the ADVISE_MQT table contains the column statistics for an MQT. The statistics are in an XML structure as follows:

```
<?xml version=\"1.0\" encoding=\"USASCII\"?>
<colstats>
    <column>
         <name>COLNAME1</name>
         <colcard>1000</colcard>
         <high2key>999</high2key>
         <low2key>2</low2key>
    </column>
    ....

    <column>
         <name>COLNAME100</name>
         <colcard>55000</colcard>
         <high2key>49999</high2key>
         <low2key>100</low2key>
    </column>
</colstats>
```

Note that the XML structure can contain more than one column. For each column, the column cardinality (that is, the number of values in the column) is shown, and optionally, the high2 and low2 keys.

When MQT, database partitions, or MDC recommendations are provided, the relevant ALTER TABLE stored procedure call command is placed in the ALTER_COMMAND column of the ADVISE_TABLE.

Note: The ALTER TABLE stored procedure call command may not succeed due to restrictions on the table for the ALTOBJ stored procedure.

After some minor modifications, you can run this output file as a CLP script to create the recommended objects. The modifications that you might want to perform include:

- Combining all of the RUNSTATS command statements into a single RUNSTATS invocation on the new or modified objects.
- Providing more usable object names than the system-generated IDs.
- Removing or commenting out any DDL for objects that you do not want to implement immediately.

Related concepts:

- "Using relational indexes to improve performance" on page 25
- "Materialized query tables" on page 11
- "Partitioned database environments" in *Administration Guide: Planning*
- "Relational index planning tips" on page 28
- "Multidimensional clustering tables" in *Administration Guide: Planning*

Related reference:

- "db2advis - DB2 design advisor command" in *Command Reference*

Design advisor features

The Design advisor considers the costs and advantages of each feature when formulating design recommendations. It takes into account the characteristics of your workload, the characteristics of the database, and the hardware resources. Then, it weighs the performance improvements against the costs associated with implementing the recommendations.

Before implementing any of the Design advisor recommendations, consider whether you provided the Design advisor with a representative query workload. If the workload that you submit to the Design advisor is not representative of the overall database activity, the recommendations might not provide performance enhancements for queries not included in the workload. Also, it is critical that the statistics on any object involved in the workload be up to date. Inaccurate statistics might negatively impact the quality of the Design advisor recommendations.

Because the Design advisor performs the cost-benefit analysis for you, all features are selected by default for consideration. It is recommended that you leave this default setting and allow the Design advisor to determine which features provide the greatest overall performance advantages for the workload that you specify.

If you do not want to implement specific performance enhancement features, you can choose not to select those features for evaluation. Eliminating those features

will reduce the amount of time that the Design advisor spends evaluating different performance enhancement possibilities for your database.

Use the following information as a guide in determining which features may not be appropriate for your system.

Disadvantages of Materialized query tables (MQTs)

MQTs incur costs for storage space and for the overhead involved in updates. The data in an MQT is stored redundantly: once in the MQT and once in the underlying base table or tables. This means that whenever the data in the base tables is updated, two updates are required for REFRESH IMMEDIATE MQTs. In a workload with many updates, this cost might outweigh the performance benefits of MQT candidates.

There will also be administrative overhead associated with MQTs whenever you run utilities such as LOAD, BACKUP, RESTORE, or RUNSTATS.

To determine whether MQTs would offer significant performance advantages, the Design advisor considers the costs associated with MQTs in conjunction with the characteristics of the workload that you specify.

Disadvantages of Multidimensional clustering (MDC) tables

Implementing any MDC table recommendations requires dropping and recreating tables as well as creating additional storage space to store the tables. If you are certain that you do not want to do this, then do not select this feature.

Disadvantages of repartitioning tables

Repartitioning tables is relevant in a shared-nothing partitioned environment only. Implementing any repartitioning recommendations requires dropping and recreating tables. If you are certain that you do not want to do this, then do not select this feature.

Related concepts:

- "About materialized query tables" in *Administration Guide: Planning*
- "Design advisor frequency" in *Administration Guide: Planning*
- "Design advisor recommendation process" in *Administration Guide: Planning*
- "Design advisor time limit" in *Administration Guide: Planning*
- "Design advisor workload" in *Administration Guide: Planning*
- "Multidimensional clustering tables" in *Administration Guide: Planning*
- "Partitioned databases" in *Administration Guide: Planning*

Using the Design Advisor

You can run the Design Advisor from the command line or by using the Design Advisor GUI in the Control Center.

Procedure:

- **Control Center method:**
  1. Define your workload..
  2. Open the Control Center.

3. Run the Design Advisor from the Control Center by right-clicking on the appropriate database or on a particular index in the database, and selecting Design Advisor from the pop-up menu.
4. Once the Design Advisor has finished generating recommendations, you can choose to save these recommendations in a report, or you can have the Design Advisor implement some or all of the recommendations for you

- **Command line method:**

1. Define your workload..
2. Run the **db2advis** command on the workload.

   Note: If the statistics on your database are not current, the generated recommendations will be less reliable.
3. Interpret the output from **db2advis** and make any necessary modifications.
4. Implement the selected Design Advisor recommendations.

Related concepts:

- "Design Advisor limitations and restrictions" on page 49
- "The Design Advisor" on page 43

Related tasks:

- "Defining a workload for the Design Advisor" on page 47
- "Using the Design Advisor to migrate from a single-partition to a multiple-partition database" on page 48

Defining a workload for the Design Advisor

A *workload* is a set of SQL statements that the database manager has to process during a given period of time. For example, during one month your database manager may have to process 1000 INSERTs, 10000 UPDATEs, 10000 SELECTs, and 1000 DELETEs. The Design Advisor analyzes a specified workload and considers factors such as the type of workload statements, the frequency with which a particular statement occurs, and characteristics of your database to generate recommendations that minimize the total cost to run the workload.

Procedure:

Design Advisor GUI method:

From the Design Advisor GUI workload page, you can create a new workload file, or modify a previously existing workload file. You can import statements into the file from several sources:

- A delimited text file
- An Event Monitor table
- Query Patroller historical data tables by using the -qp option from the command line
- Explained statements in the EXPLAINED_STATEMENT table
- Recent SQL statements that have been captured with a DB2 snapshot.

After you import your SQL statements, you can add, change, modify, or remove statements and modify their frequency.

Design Advisor command-line method:

From the command line, run the Design Advisor using:

- A single SQL statement that you enter in-line with the command
- A set of dynamic SQL statements captured in a DB2 snapshot
- A set of SQL statements contained in a workload file.

To run the Design Advisor on dynamic SQL statements:

1. Reset the database monitor with the following command:

   ```
   db2 reset monitor for database database-name
   ```

2. Wait for an appropriate interval of time for the execution of dynamic SQL statements against the database.
3. Issue the **db2advis** command with the -g option. If you want to save the dynamic SQL statements in the ADVISE_WORKLOAD table for later reference, use the -p option as well.

To run the Design Advisor on a set of SQL statements contained in a workload file:

1. Create a workload file manually, separating each SQL statement with a semicolon, or import SQL statements from one or more of the sources listed above.
2. Set the frequency of the statements in the workload. Every statement in the workload file is assigned a frequency of 1 by default. The frequency of an SQL statement represents the number of times the statement occurs within a workload relative to the number of times that other statements occur. For example, a particular SELECT statement might occur 100 times in a workload, while another SELECT statement occurs 10 times. To represent the relative frequency of these two statements, you could assign the first SELECT statement a frequency of 10, while the second select statement has a frequency of 1. You can manually change the frequency or weight that a particular statement has in the workload by inserting the following line after the statement - - # SET FREQUENCY *n* where *n* is the frequency value that you want to assign to the statement.
3. Run the **db2advis** command using the -i option followed by the name of the workload file.

To run the Design Advisor on a workload contained in the ADVISE_WORKLOAD table, run **db2advis** with the -w option followed by the workload name.

Related concepts:

- "The Design Advisor" on page 43

Using the Design Advisor to migrate from a single-partition to a multiple-partition database

You can use the Design Advisor to help you migrate from a single-partition to a multiple-partition database. The Design Advisor can provide you with recommendations for distributing your data and, at the same time, provide you with recommendations for new indexes, materialized query tables (MQTs), and multi-dimensional clustering (MDC) tables.

Procedure:

1. Update the product license key for DB2 ESE.
2. Create at least one table space in a multiple-partition database partition group.

Note: Because the Design Advisor can only recommend data redistribution to existing table spaces, the table spaces that you want the Design Advisor to consider must exist in the partitioned database before the Design Advisor is run.

3. Run the Design Advisor, with the database partitioning feature selected in the Design Advisor GUI, or with the partitioning option specified for the **db2advis** command.
4. If you are using the Design Advisor in the Control Center, you can implement the database partitioning recommendations automatically. If you are using the **db2advis** command you will need to modify the **db2advis** output file slightly before running the DDL statements generated by the Design Advisor.

Related concepts:

- "The Design Advisor" on page 43

Related tasks:

- "Registering a DB2 product or feature license key using the db2licm command" in *Installation and Configuration Supplement*

Design Advisor limitations and restrictions

1. **Restrictions on index recommendations**
   - Indexes recommended on materialized query tables (MQTs) are to improve the workload performance as opposed to REFRESH TABLE performance. Also if updates, inserts, or deletes are not included in the workload, the performance of changing (for example, updating) the MQT would not be included for IMMEDIATE MQTs.
   - The clustering RID index is only recommended when multidimensional clustering is to be selected. The advisor will include RID clustering indexes as an option instead of creating an MDC structure for a table.
2. **Restrictions on MQT recommendations**
   - The Design Advisor will not recommend incremental MQTs. If you want to create incremental MQTs, you can take REFRESH IMMEDIATE MQTs and convert these to incrementals with your choice of staging tables.
   - Indexes recommended for MQTs are designed to improve workload performance and not MQT refresh performance.
   - If updates, inserts, or deletes are not included in the specified workload, the performance impact of updating a recommended REFRESH IMMEDIATE MQT is not considered. It is recommended that REFRESH IMMEDIATE MQTs have unique indexes created on the implied unique key, which is composed of the columns in the GROUP BY clause of the MQT query definition.
3. **Restrictions on MDC recommendations**
   - Existing tables must have data in them; otherwise, MDC will not be considered for the table.
   - MDC recommendations for new MQTs will not be considered unless the sampling option, -r, is used with the command, or MQT sampling is selected in the GUI tool.
   - The Design Advisor does not make MDC recommendations for typed or temporary tables.
   - The Design Advisor does not make MDC recommendations for federated tables.

- Storage space must exist for the sampling data used during the execution of the Design Advisor, otherwise the sampled table will be examined only for base columns under the uncorrelated assumption. A warning message will be generated in this case.
- Tables without statistics collected will be skipped for consideration.
- The Design Advisor does not make recommendations for multicolumn dimensions.
- Existing tables must have data in them for sampling to work in MDC selection.

4. **Restrictions on database partitioning recommendations**

   The Design Advisor can only recommend database partitioning on DB2 Enterprise Server Edition. If the database partitioning options are specified with the **db2advis** command, an error is returned. In the Design Advisor GUI, the database partitioning feature is not selectable in a single-partition database environment.

5. **Additional restrictions**

   Simulation catalog tables are created during the execution of the Design Advisor. These tables are dropped when the Design Advisor execution completes. Incomplete Design Advisor execution may result in some of these tables not being dropped. In this situation, you can use the Design Advisor to drop these tables by restarting the utility from the command line. To remove the simulation catalog tables, specify both the -f option and the -n option (for -n, specifying the same user name that was used for the incomplete execution). If you do not specify the -f option, the Design Advisor will generate the DROP statements that are required to remove the tables.

   You should create a separate table space for storing these simulated catalog tables and set DROP TABLE RECOVERY to ″OFF″. This will allow for easier cleanup and for faster Design Advisor execution.

Related concepts:

• "The Design Advisor" on page 43

Chapter 5. Managing concurrency

Concurrency issues

Because many users access and change data in a relational database, the database manager must be able both to allow users to make these changes and to ensure that data integrity is preserved. *Concurrency* refers to the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

- **Lost updates**. Two applications, A and B, might both read the same row from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.
- **Access to uncommitted data**. Application A might update a value in the database, and application B might read that value before it was committed. Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.
- **Nonrepeatable reads**. Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other requests. In the meantime, application B either modifies or deletes the row and commits the change. Later, if application A attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.
- **Phantom Read Phenomenon**. The phantom read phenomenon occurs when:
  1. Your application executes a query that reads a set of rows based on some search criterion.
  2. Another application inserts new data or updates existing data that would satisfy your application's query.
  3. Your application repeats the query from step 1 (within the same unit of work).

     Some additional ("phantom") rows are returned as part of the result set that were not returned when the query was initially executed (step 1).

Note: Declared temporary tables have no concurrency issues because they are available only to the application that declared them. This type of table only exists from the time that the application declares it until the application completes or disconnects.

Concurrency control in federated database systems

A *federated database system* supports applications and users submitting SQL statements that reference two or more database management systems (DBMSs) or databases in a single statement. To reference the data sources, which consist of a DBMS and data, DB2 uses *nicknames*. Nicknames are aliases for objects in other database managers. In a federated system, DB2 relies on the concurrency control protocols of the database manager that hosts the requested data.

A DB2 federated system provides *location transparency* for database objects. For example, with location transparency if information about tables and views is moved, references to that information through nicknames can be updated without changing applications that request the information. When an application accesses

data through nicknames, DB2 relies on the concurrency control protocols of data-source database managers to ensure isolation levels. Although DB2 tries to match the requested level of isolation at the data source with a logical equivalent, results may vary depending on data source capabilities.

Related concepts:

- "Isolation levels and performance" on page 52

Related tasks:

- "Specifying the isolation level" on page 55

Related reference:

- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "locklist - Maximum storage for lock list " on page 383

Isolation levels

Isolation levels and performance

An *isolation level* determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. Applications that use a cursor declared with a DECLARE CURSOR statement using the WITH HOLD clause will keep the chosen isolation level for the duration of the unit of work in which the OPEN CURSOR was performed. DB2 supports the following isolation levels:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

Note: Some host database servers support the *no commit* isolation level. On other databases, this isolation level behaves like the uncommitted read isolation level.

Detailed explanations for each of the isolation levels follows in decreasing order of performance impact, but in increasing order of care required when accessing and updating data.

Repeatable Read

Repeatable Read (RR) locks all the rows an application references within a unit of work. Using Repeatable Read, a SELECT statement issued by an application twice within the same unit of work in which the cursor was opened returns the same result each time. With Repeatable Read, lost updates, access to uncommitted data, and phantom rows are not possible.

The Repeatable Read application can retrieve and operate on the rows as many times as needed until the unit of work completes. However, no other applications can update, delete, or insert a row that would affect the result table, until the unit of work completes. Repeatable Read applications cannot see uncommitted changes of other applications.

With Repeatable Read, every row that is referenced is locked, not just the rows that are retrieved. Appropriate locking is performed so that another application cannot

insert or update a row that would be added to the list of rows referenced by a query if that query were to be re-executed. This prevents phantom rows from occurring. For example, if you scan 10 000 rows and apply predicates to them, locks are held on all 10 000 rows, even though only 10 rows qualify.

Note: The Repeatable Read isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking are used.

Since Repeatable Read may acquire and hold a considerable number of locks, these locks may exceed the number of locks available as a result of the *locklist* and *maxlocks* configuration parameters. In order to avoid lock escalation, the optimizer may elect to acquire a single table-level lock immediately for an index scan, if it believes that lock escalation is very likely to occur. This functions as though the database manager has issued a LOCK TABLE statement on your behalf. If you do not want a table-level lock to be obtained ensure that enough locks are available to the transaction or use the Read Stability isolation level.

When evaluating referential constraints, there are a few situations where DB2 will internally upgrade the isolation level used on the scan on the child table to Repeatable Read (RR), regardless of the isolation level set by the user. This will result in additional locks being held until commit, which increases the likelihood of a deadlock or lock timeout occurring. To avoid this, it is recommended that you create an index that only contains the column or columns of the foreign key, allowing the RI scan to use this index instead.

Read Stability

Read Stability (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, "nonrepeatable read" behavior is **not** possible.

Unlike repeatable read, with Read Stability, if your application issues the same query more than once, you may see additional *phantom* rows (the *phantom read phenomenon*). Recalling the example of scanning 10 000 rows, Read Stability only locks the rows that qualify. Thus, with Read Stability, only 10 rows are retrieved, and a lock is held only on those ten rows. Contrast this with Repeatable Read, where in this example, locks would be held on all 10 000 rows. The locks that are held can be share, next share, update, or exclusive locks.

Note: The Read Stability isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

One of the objectives of the Read Stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs.

The Read Stability isolation level is best for applications that include all of the following:
- Operate in a concurrent environment
- Require qualifying rows to remain stable for the duration of the unit of work

- Do not issue the same query more than once within the unit of work, or do not require that the query get the same answer when issued more than once in the same unit of work.

Cursor Stability

Cursor Stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a Cursor Stability application has retrieved while any updatable cursor is positioned on the row. Cursor Stability applications cannot see uncommitted changes of other applications.

Recalling the example of scanning 10 000 rows, if you use Cursor Stability, you will only have a lock on the row under your current cursor position. The lock is removed when the cursor moves off that row (unless you update that row).

With Cursor Stability, both nonrepeatable read and the phantom read phenomenon are possible. Cursor Stability is the default isolation level and should be used when you want the maximum concurrency while seeing only committed rows from other applications.

Uncommitted Read

Uncommitted Read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted Read works differently for read-only and updatable cursors.

Read-only cursors can access most uncommitted changes of other transactions. However, tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back.

Note: Cursors that are updatable operating under the Uncommitted Read isolation level will behave as if the isolation level was cursor stability.

When it runs a program using isolation level UR, an application can use isolation level CS. This happens because the cursors used in the application program are ambiguous. The ambiguous cursors can be escalated to isolation level CS because of a BLOCKING option. The default for the BLOCKING option is UNAMBIG. This means that ambiguous cursors are treated as updatable and the escalation of the isolation level to CS occurs. To prevent this escalation, you have the following two choices:

- Modify the cursors in the application program so that they are unambiguous. Change the SELECT statements to include the FOR READ ONLY clause.
- Leave cursors ambiguous in the application program, but precompile the program or bind it with the BLOCKING ALL option to allow any ambiguous cursors to be treated as read-only when the program is run.

As in the example given for Repeatable Read, of scanning 10 000 rows, if you use Uncommitted Read, you do not acquire any row locks.

With Uncommitted Read, both nonrepeatable read behavior and the phantom read phenomenon are possible. The Uncommitted Read isolation level is most commonly used for queries on read-only tables, or if you are executing only select statements and you do not care whether you see uncommitted data from other applications.

Summary of isolation levels

The following table summarizes the different isolation levels in terms of their undesirable effects.

Table 2. Summary of isolation levels

Isolation Level	Access to uncommitted data	Nonrepeatable reads	Phantom read phenomenon
Repeatable Read (RR)	Not possible	Not possible	Not possible
Read Stability (RS)	Not possible	Not possible	Possible
Cursor Stability (CS)	Not possible	Possible	Possible
Uncommitted Read (UR)	Possible	Possible	Possible

The table below provides a simple heuristic to help you choose an initial isolation level for your applications. Consider this table a starting point, and refer to the previous discussions of the various levels for factors that might make another isolation level more appropriate.

Table 3. Guidelines for choosing an isolation level

Application Type	High data stability required	High data stability not required
Read-write transactions	RS	CS
Read-only transactions	RR or RS	UR

Choosing the appropriate isolation level for an application is very important to avoid the phenomena that are intolerable for that application. The isolation level affects not only the degree of isolation among applications but also the performance characteristics of an individual application since the CPU and memory resources that are required to obtain and free locks vary with the isolation level. The potential for deadlock situations also varies with the isolation level.

Related concepts:

- "Concurrency issues" on page 51

Related tasks:

- "Specifying the isolation level" on page 55

Specifying the isolation level

Because the isolation level determines how data is locked and isolated from other processes while the data is being accessed, you should select an isolation level that

balances the requirements of concurrency and data integrity. The isolation level that you specify is in effect for the duration of the unit of work.

Note: Isolation levels cannot be specified for XQuery statements at the statement level.

The isolation level can be specified in several different ways. The following heuristics are used in determining which isolation level will be used in compiling an SQL or XQuery statement:

Static SQL:

- If an isolation clause is specified in the statement, then the value of that clause is used.
- If no isolation clause is specified in the statement, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Dynamic SQL:

- If an isolation clause is specified in the statement, then the value of that clause is used.
- If no isolation clause is specified in the statement, and a SET CURRENT ISOLATION statement has been issued within the current session, then the value of the CURRENT ISOLATION special register is used.
- If no isolation clause is specified in the statement, and no SET CURRENT ISOLATION statement has been issued within the current session, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Static or dynamic XQuery statements:

- The isolation level of the environment determines the isolation level when the XQuery expression is evaluated.

Note: Many commercially written applications provide a method for choosing the isolation level. Refer to the application documentation for information.

Procedure:

To specify the isolation level:

1. **At precompile or bind time:**

   For an application written in a supported compiled language, use the ISOLATION option of the command line processor PREP or BIND commands. You can also use the PREP or BIND APIs to specify the isolation level.

   - If you create a bind file at precompile time, the isolation level is stored in the bind file. If you do not specify an isolation level at bind time, the default is the isolation level used during precompilation.
   - If you do not specify an isolation level, the default of cursor stability is used.

   Note: To determine the isolation level of a package, execute the following query:

   ```
   SELECT ISOLATION FROM SYSCAT.PACKAGES
     WHERE PKGNAME = 'XXXXXXXX'
     AND PKGSCHEMA = 'YYYYYYYY'
   ```

where *XXXXXXXX* is the name of the package and *YYYYYYYY* is the schema name of the package. Both of these names must be in all capital letters.

2. **On database servers that support REXX:**

   When a database is created, multiple bind files that support the different isolation levels for SQL in REXX are bound to the database. Other command-line processor packages are also bound to the database when a database is created.

   REXX and the command line processor connect to a database using a default isolation level of cursor stability. Changing to a different isolation level does not change the connection state. It must be executed in the CONNECTABLE AND UNCONNECTED state or in the IMPLICITLY CONNECTABLE state.

   To verify the isolation level in use by a REXX application, check the value of the SQLISL REXX variable. The value is updated every time the CHANGE SQLISL command is executed.

3. **At the statement level:**

   Use the WITH clause. The statement-level isolation level overrides the isolation level specified for the package in which the statement appears.

   You can specify an isolation level for the following SQL statements:

   - SELECT
   - SELECT INTO
   - Searched DELETE
   - INSERT
   - Searched UPDATE
   - DECLARE CURSOR

   The following conditions apply to isolation levels specified for statements:

   - The WITH clause cannot be used on subqueries
   - The WITH UR option applies only to read-only operations. In other cases, the statement is automatically changed from UR to CS.

4. **From CLI or ODBC at runtime:**

   Use the CHANGE ISOLATION LEVEL command. For DB2 Call Level Interface (DB2 CLI), you can change the isolation level as part of the DB2 CLI configuration. At runtime, use the *SQLSetConnectAttr* function with the SQL_ATTR_TXN_ISOLATION attribute to set the transaction isolation level for the current connection referenced by the *ConnectionHandle*. You can also use the TXNISOLATION keyword in the db2cli.ini file .

5. **When working with JDBC or SQLJ at run time:**

   Note: JDBC and SQLJ are implemented with CLI on DB2, which means the db2cli.ini settings might affect what is written and run using JDBC and SQLJ.

   In SQLJ, you use the SQLJ profile customizer (db2sqljcustomize command) to create a package. The options that you can specify for this package include its isolation level.

6. **For dynamic SQL within the current session:**

   Use the SET CURRENT ISOLATION statement to set the isolation level for dynamic SQL issued within a session. Issuing this statement sets the CURRENT ISOLATION special register to a value that specifies the level of isolation for any dynamic SQL statements issued within the current session. Once set, the CURRENT ISOLATION special register provides the isolation level for any subsequent dynamic SQL statement compiled within the session, regardless of

the package issuing the statement. This isolation level will apply until the session is ended or until a SET CURRENT ISOLATION statement is issued with the RESET option.

Related concepts:

- "Concurrency issues" on page 51
- "Isolation levels" in *SQL Reference, Volume 1*

Related reference:

- "CONNECT (Type 1) statement" in *SQL Reference, Volume 2*
- "SQLSetConnectAttr function (CLI) - Set connection attributes" in *Call Level Interface Guide and Reference, Volume 2*
- "Statement attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*

Locking

Locks and concurrency control

To provide concurrency control and prevent uncontrolled data access, the database manager places locks on buffer pools, tables, data partitions, table blocks, or table rows. A *lock* associates a database manager resource with an application, called the *lock owner*, to control how other applications access the same resource.

The database manager uses row-level locking or table-level locking as appropriate based on:

- The isolation level specified at precompile time or when an application is bound to the database. The isolation level can be one of the following:
  - Uncommitted Read (UR)
  - Cursor Stability (CS)
  - Read Stability (RS)
  - Repeatable Read (RR)

  The different isolation levels are used to control access to uncommitted data, prevent lost updates, allow non-repeatable reads of data, and prevent phantom reads. To minimize performance impact, use the minimum isolation level that satisfies your application needs.
- The access plan selected by the optimizer. Table scans, index scans, and other methods of data access each require different types of access to the data.
- The LOCKSIZE attribute for the table. The LOCKSIZE clause on the ALTER TABLE statement indicates the granularity of the locks used when the table is accessed. The choices are either ROW for row locks, TABLE for table locks, or BLOCKINSERT for block locks on MDC tables only. When the BLOCKINSERT clause is used on an MDC table, row-level locking is performed except on an INSERT operation where block-level locking is done instead. Use the ALTER TABLE ... LOCKSIZE BLOCKINSERT statement for MDC tables when transactions perform large inserts into disjoint cells. Use the ALTER TABLE ... LOCKSIZE TABLE statement for read-only tables. This reduces the number of locks required by database activity. For partitioned tables, table locks are first acquired and then data partition locks are acquired, as dictated by the data accessed.

- The amount of memory devoted to locking. The amount of memory devoted to locking is controlled by the `locklist` database configuration parameter. If the lock list fills, performance can degrade due to lock escalations and reduced concurrency on shared objects in the database. If lock escalations occur frequently, increase the value of either `locklist` or `maxlocks`, or both. Also, to reduce number of locks held at one time, ensure that transactions COMMIT frequently to free held locks.

Although most locking occurs on tables, when a buffer pool is created, altered, or dropped, a buffer pool lock is set. The mode used with this lock is EXCLUSIVE (X). You may encounter this type of lock when collecting system monitoring data. When viewing the snapshot, you will see that the lock name used is the identifier (ID) of the buffer pool itself.

In general, row-level locking is used unless one of the following is the case:

- The isolation level chosen is uncommitted read (UR).
- The isolation level chosen is repeatable read (RR) and the access plan requires a scan with no predicates.
- The table LOCKSIZE attribute is "TABLE".
- The lock list fills, causing escalation.
- There is an explicit table lock acquired via the LOCK TABLE statement. The LOCK TABLE statement prevents concurrent application processes from either changing a table or using a table.

If this is an MDC table, there are several other cases where block-level locking is used instead of row-level locking, including:

- The table LOCKSIZE attribute is "BLOCKINSERT"
- The isolation level chosen is repeatable read (RR) and the access plan involves predicates
- A searched update or delete operation that involves only predicates on dimension columns

Lock escalations reduce concurrency. Conditions that might cause lock escalations should be avoided.

The duration of row locking varies with the isolation level being used:

- UR scans: No row locks are held unless row data is changing.
- CS scans: Row locks are only held while the cursor is positioned on the row.
- RS scans: Only qualifying row locks are held for the duration of the transaction.
- RR scans: All row locks are held for the duration of the transaction.

Related concepts:

- "Preventing lock-related performance issues" on page 69
- "Lock attributes" on page 60
- "Lock granularity" on page 62

Related reference:

- "Lock modes for table and RID index scans of MDC tables" on page 81
- "Lock type compatibility" on page 71
- "Lock modes and access paths for standard tables" on page 74
- "Locking for block index scans for MDC tables" on page 85

- "diaglevel - Diagnostic error capture level " on page 493
- "locktimeout - Lock timeout " on page 413
- "locklist - Maximum storage for lock list " on page 383
- "dlchktime - Time interval for checking deadlock " on page 413

Lock attributes

Database manager locks have the following basic attributes:

Mode The type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked object. It is sometimes referred to as the *state* of the lock.

Object
The resource being locked. The only type of object that you can lock explicitly is a table. The database manager also imposes locks on other types of resources, such as rows, tables, and table spaces. For multidimensional clustering (MDC) tables, block locks can also be imposed; for partitioned tables, data partition locks can be imposed. The object being locked determines the *granularity* of the lock.

Duration
The length of time a lock is held. The isolation level in which the query runs affects the lock duration.

The following table shows the modes and their effects in order of increasing control over resources. For detailed information about locks at various levels, refer to the lock-mode reference tables.

Table 4. Lock Mode Summary

Lock Mode	Applicable Object Type	Description
IN (Intent None)	Table spaces, blocks, tables, data partitions	The lock owner can read any data in the object, including uncommitted data, but cannot update any of it. Other concurrent applications can read or update the table.
IS (Intent Share)	Table spaces, blocks, tables, data partitions	The lock owner can read data in the locked table, but cannot update this data. Other applications can read or update the table.
NS (Next Key Share)	Rows	The lock owner and all concurrent applications can read, but not update, the locked row. This lock is acquired on rows of a table, instead of an S lock, where the isolation level of the application is either RS or CS. NS lock mode is not used for next-key locking. It is used instead of S mode during CS and RS scans to minimize the impact of next-key locking on these scans.
S (Share)	Rows, blocks, tables, data partitions	The lock owner and all concurrent applications can read, but not update, the locked data.
IX (Intent Exclusive)	Table spaces, blocks, tables, data partitions	The lock owner and concurrent applications can read and update data. Other concurrent applications can both read and update the table.
SIX (Share with Intent Exclusive)	Tables, blocks, data partitions	The lock owner can read and update data. Other concurrent applications can read the table.
U (Update)	Rows, blocks, tables, data partitions	The lock owner can update data. Other units of work can read the data in the locked object, but cannot attempt to update it.

Table 4. Lock Mode Summary (continued)

Lock Mode	Applicable Object Type	Description
NW (Next Key Weak Exclusive)	Rows	When a row is inserted into an index, an NW lock is acquired on the next row. For type 2 indexes, this occurs only if the next row is currently locked by an RR scan. The lock owner can read but not update the locked row. This lock mode is similar to an X lock, except that it is also compatible with W and NS locks.
X (Exclusive)	Rows, blocks, tables, buffer pools, data partitions	The lock owner can both read and update data in the locked object. Only uncommitted read applications can access the locked object.
W (Weak Exclusive)	Rows	This lock is acquired on the row when a row is inserted into a table that does not have type-2 indexes defined. The lock owner can change the locked row. To determine if a duplicate value has been committed when a duplicate value is found, this lock is also used during insertion into a unique index. This lock is similar to an X lock except that it is compatible with the NW lock. Only uncommitted read applications can access the locked row.
Z (Super Exclusive)	Table spaces, tables, data partitions	This lock is acquired on a table in certain conditions, such as when the table is altered or dropped, an index on the table is created or dropped, or for some types of table reorganization. No other concurrent application can read or update the table.

Related concepts:

- "Locks and concurrency control" on page 58
- "Lock granularity" on page 62

Related reference:

- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "Lock type compatibility" on page 71
- "Lock modes and access paths for standard tables" on page 74

Locks and types of application processing

For the purpose of determining lock attributes, application processing can be classified as one of the following types:

- Read-only

  This type includes all select statements that are intrinsically read-only, have an explicit FOR READ ONLY clause, or are ambiguous but which the query compiler assumes to be read-only because of the value of the BLOCKING option that the PREP or BIND command specifies. This processing type requires only Share locks (S, NS, or IS).

- Intent to change

  This type includes all select statements with the FOR UPDATE clause, with the USE AND KEEP UPDATE LOCKS clause, with the USE AND KEEP EXCLUSIVE LOCKS clause, or for which the query compiler interprets an ambiguous statement to imply that change is intended. This type uses Share and Update locks (S, U, and X for rows; IX, U, X, and S for blocks; IX, U, and X for tables).

- Change

  This type includes UPDATE, INSERT, and DELETE, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. This type requires Exclusive locks (X or IX).

- Cursor controlled

  This type includes UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF. It also requires Exclusive locks (X or IX).

A statement that inserts, updates or deletes data in a target table, based on the result from a sub-select statement, does two types of processing. The rules for read-only processing determine the locks for the tables returned in the sub-select statement. The rules for change processing determine the locks for the target table.

Related concepts:

- "Deadlocks" on page 65
- "Preventing lock-related performance issues" on page 69
- "Index types and next-key locking" on page 77
- "Lock attributes" on page 60
- "Locks and concurrency control" on page 58
- "Locks and data-access methods" on page 72
- "Lock granularity" on page 62

Related tasks:

- "Correcting lock escalation problems" on page 64

Related reference:

- "Lock type compatibility" on page 71

Lock granularity

If one application holds a lock on a database object, another application might not be able to access that object. For this reason, row-level locks, which minimize the amount of data that is locked and therefore inaccessible, are better for maximum concurrency than block-level, data partition-level or table-level locks. However, locks require storage and processing time, so a single table lock minimizes lock overhead.

The LOCKSIZE clause of the ALTER TABLE statement specifies the scope (granularity) of locks at the row, data partition, block, or table level. By default, row locks are used. Only S (Shared) and X (Exclusive) locks are requested by these defined table locks. The ALTER TABLE statement LOCKSIZE ROW clause does not prevent normal lock escalation from occurring.

A permanent table lock defined by the ALTER TABLE statement might be preferable to a single-transaction table lock using LOCK TABLE statement in the following cases:

- The table is read-only, and will always need only S locks. Other users can also obtain S locks on the table.
- The table is usually accessed by read-only applications, but is sometimes accessed by a single user for brief maintenance, and that user requires an X lock. While the maintenance program runs, the read-only applications are locked out, but in other circumstances, read-only applications can access the table concurrently with a minimum of locking overhead.

For an MDC table, you can specify BLOCKINSERT for the LOCKSIZE clause in order to use block-level locking during INSERT operations only. When this is specified, row-level locking is performed for all other operations, but only

minimally for INSERT operations. That is, block-level locking is used during the insertion of rows, but row-level locking is used for next-key locking if RR scans are encountered in the indexes as they are being updated. BLOCKINSERT locking might be beneficial in the following cases:

- There are multiple transactions doing mass insertions into separate cells.
- Concurrent insertions to the same cell by multiple transactions is not occurring, or it is occurring with enough data inserted per cell by each of the transactions that the user is not concerned that each transaction will insert into separate blocks.

The ALTER TABLE statement specifies locks globally, affecting all applications and users that access that table. Individual applications might use the LOCK TABLE statement to specify table locks at an application level instead.

Related concepts:

- "Deadlocks" on page 65
- "Locks and concurrency control" on page 58

Related tasks:

- "Correcting lock escalation problems" on page 64

Related reference:

- "diaglevel - Diagnostic error capture level " on page 493
- "locktimeout - Lock timeout " on page 413

Lock conversion

Changing the mode of a lock already held is called a *conversion*. Lock conversion occurs when a process accesses a data object on which it already holds a lock, and the access mode requires a more restrictive lock than the one already held. A process can hold only one lock on a data object at any time, although it can request a lock many times on the same data object indirectly through a query.

Some lock modes apply only to tables, others only to rows or blocks. For rows or blocks, conversion usually occurs if an X is needed and an S or U (Update) lock is held.

IX (Intent Exclusive) and S (Shared) locks are special cases with regard to lock conversion, however. Neither S nor IX is considered to be more restrictive than the other, so if one of these is held and the other is required, the resulting conversion is to a SIX (Share with Intent Exclusive) lock. All other conversions result in the requested lock mode becoming the mode of the lock held if the requested mode is more restrictive.

A dual conversion might also occur when a query updates a row. If the row is read through an index access and locked as S, the table that contains the row has a covering intention lock. But if the lock type is IS instead of IX and the row is subsequently changed, the table lock is converted to an IX and the row to an X.

Lock conversion usually takes place implicitly as a query is executed. Understanding the kinds of locks obtained for different queries and table and index combinations can assist you in designing and tuning your application.

The system monitor elements *lock_current_mode* and *lock_mode* can provide information about lock conversions occurring in your database.

Related concepts:

- "Lock attributes" on page 60
- "Locks and concurrency control" on page 58

Related reference:

- "lock_current_mode - Original Lock Mode Before Conversion monitor element" in *System Monitor Guide and Reference*
- "lock_mode - Lock Mode monitor element" in *System Monitor Guide and Reference*
- "Lock type compatibility" on page 71

Lock escalation

A lock escalation occurs when the number of locks held on rows and tables in the database equals the percentage of the lock list specified by the `maxlocks` database configuration parameter. Lock escalation might not affect the table that acquires the lock triggering the escalation. To reduce the number of locks to about half the number held when the lock escalation began, the database manager begins converting many small row locks to table locks for all active tables, beginning with any locks on large object (LOB) or long VARCHAR elements. An *exclusive lock escalation* is a lock escalation in which the table lock acquired is an exclusive lock.

For partitioned tables, lock escalation is to the data partition level. This allows the table to be accessible to other transactions even if the data partition is escalated to share, exclusive, or super exclusive, as other non-escalated data partitions are unaffected. The transaction may continue to row lock on other data partitions after escalation for a given data partition. For partitioned tables, the notification log messages for escalations will include the data partition escalated as well as the table name.

Correcting lock escalation problems

The database manager can automatically escalate locks from row or block level to table level. For partitioned tables, the database manager can automatically escalate locks from row or block level to data partition level. The *maxlocks* database configuration parameter specifies when lock escalation is triggered. The table that acquires the lock that triggers lock escalation might not be affected. Locks are first escalated for the table with the most locks, beginning with tables for which long object (LOBs) and long VARCHAR descriptors are locked, then the table with the next highest number of locks, and so on, until the number of locks held is decreased to about half of the value specified by *maxlocks*.

In a well designed database, lock escalation rarely occurs. If lock escalation reduces concurrency to an unacceptable level (indicated by the *lock_escalation* monitor element or the *db.lock_escal_rate* health indicator) you need to analyze the problem and decide how to solve it.

Prerequisites:

Ensure that lock escalation information is recorded. Set the database manager configuration parameter *notifylevel* to 3, which is the default, or to 4. At *notifylevel* of 2, only the error SQLCODE is reported. At *notifylevel* of 3 or 4, when lock escalation fails, information is recorded for the error SQLCODE and the table for

which the escalation failed. The current query statement is logged only if it is a currently executing, dynamic query statement and *notifylevel*is set to 4.

Procedure:

Follow these general steps to diagnose the cause of unacceptable lock escalations and apply a remedy:

1. Analyze in the administration notification log on all tables for which locks are escalated. This log file includes the following information:
   - The number of locks currently held.
   - The number of locks needed before lock escalation is completed.
   - The table identifier information and table name of each table being escalated.
   - The number of non-table locks currently held.
   - The new table level lock to be acquired as part of the escalation. Usually, an "S," or Share lock, or an "X," or eXclusive lock is acquired.
   - The internal return code of the result of the acquisition of the new table lock level.
2. Use the information in administration notification log to decide how to resolve the escalation problem. Consider the following possibilities:
   - Increase the number of locks allowed globally by increasing the value of the *maxlocks* or the *locklist* parameters, or both, in the database configuration file. In a partitioned database, make this change on all database partitions.

     You might choose this method if concurrent access to the table by other processes is most important. However, the overhead of obtaining record level locks can induce more delay to other processes than is saved by concurrent access to a table.
   - Adjust the process or processes that caused the escalation. For these processes, you might issue LOCK TABLE statements explicitly.
   - Change the degree of isolation. Note that this may lead to decreased concurrency, however.
   - Increase the frequency of commits to reduce the number of locks held at a given time.
   - Consider frequent COMMIT statements for transactions that require long VARCHAR or various kinds of long object (LOB) data. Although this kind of data is not retrieved from disk until the result set is materialized, the descriptor is locked when the data is first referenced. As a result, many more locks might be held than for rows that contain more ordinary kinds of data.

Related reference:

- "diaglevel - Diagnostic error capture level " on page 493
- "maxlocks - Maximum percent of lock list before escalation " on page 414

Deadlocks

A deadlock is created when two applications are each locking data needed by the other, resulting in a situation when neither application can continue execution. For example, in the following diagram, there are two applications are running concurrently: Application A and Application B. The first step of application A is to update the first row of Table 1, and the second step is to update the second row of Table 2. Application B updates the second row of Table 2 first, and then the first row of Table 1. At one point in time, T1, Application A is executing its first step, locking the first row of Table 1 to update it. At the same time, Application B locks

the second row in Table 2 to make an update. At T2, Application A tries to execute the next step and requests a lock on the second row in Table 2 for an update. However, at the same time, Application B is trying to lock and update the first row in Table 1. Since Application A will not release its lock on the first row of Table 1 until it is able to complete an update of the second row in Table 2, and Application B will not release its lock on the second row on Table 2 until it can lock and update the first row of Table 1, a deadlock occurs. The applications can wait forever until one application releases the lock on the held data.

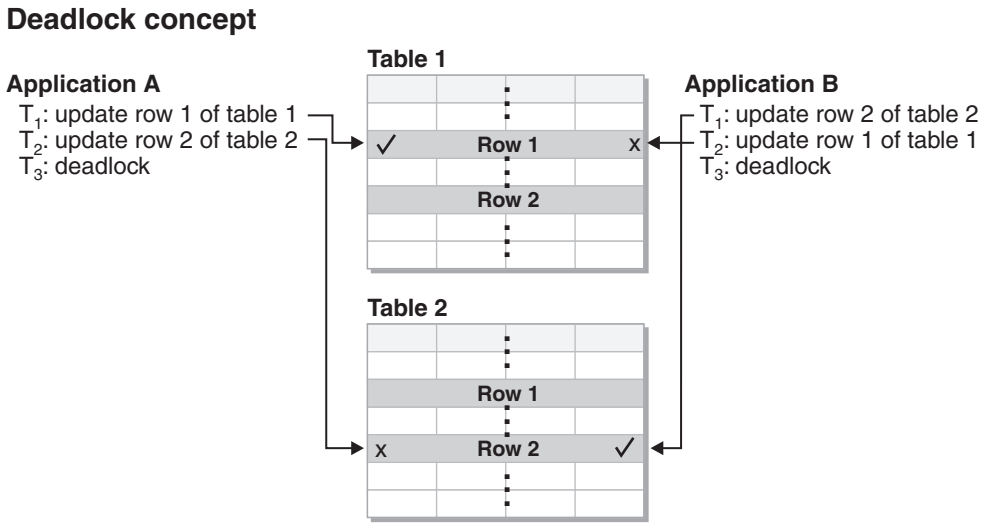


Figure 10. Deadlock between applications

Because applications do not voluntarily release locks on data that they need, a deadlock detector process is required to break deadlocks and allow application processing to continue. As its name suggests, the deadlock detector monitors the information about agents waiting on locks, awakening at intervals specified by the *dlchktime* configuration parameter.

If it finds a deadlock, the deadlock detector arbitrarily selects one deadlocked process as the *victim process* to roll back. The victim process is awakened, and returns SQLCODE -911 (SQLSTATE 40001), with reason code 2, to the calling application. The database manager rolls back the selected process automatically. When the rollback is complete, the locks that belonged to the victim process are released, and the other processes involved in the deadlock can continue.

To ensure good performance, select the proper interval for the deadlock detector. An interval that is too short causes unnecessary overhead, and an interval that is too long allows a deadlock to delay a process for an unacceptable amount of time. For example, a wake-up interval of 5 minutes may allow a deadlock to exist for almost 5 minutes, which can seem like a long time for short transaction processing. It is important to balance the possible delays in resolving deadlocks with the overhead of detecting them.

Notes:

1. In a partitioned database environment, the *dlchktime* configuration parameter interval is applied only at the catalog node. If a large number of deadlocks are detected in a partitioned database environment, increase the value of the *dlchktime* parameter to account for lock waits and communication waits.

2. In a partitioned database, each database partition sends *lock graphs* to the database partition that contains the system catalog views. Global deadlock detection takes place on this database partition.

A different problem occurs when an application with more than one independent process that accesses the database is structured to make deadlocks likely. For example, an application in which several processes access the same table for reads and then writes. If the processes do read-only SQL or XQuery queries and then do SQL updates on the same table, the chance of deadlocks increases because of potential contention between the processes for the same data. For instance, if two processes read the table and then update the table, process A might try to get an X lock on a row on which process B has an S lock. To avoid such deadlocks, applications that access data with the intention of modifying it should do one of the following:

- Use the FOR UPDATE OF clause when performing a select operation. This clause ensures that a U lock is imposed when process A attempts to read the data. Row blocking is disabled.
- Use the WITH RR USE AND KEEP UPDATE LOCKS clause or the WITH RS USE AND KEEP UPDATE LOCKS clause when performing the query. Either clause ensures that a U lock is imposed when process A attempts to read the data and allows row blocking.

At the same time a database is created, a detailed deadlocks event monitor is created. As with any monitor, there is some overhead associated with this event monitor.

To limit the amount of disk space that this event monitor consumes, the event monitor deactivates and a message is written to the administration notification log when it has reached its maximum number of output files. Removing output files that are no longer needed allows the event monitor to reactivate on the next database activation.

If you do not want the detailed deadlocks event monitor, the event monitor can be dropped using the command:

```
DROP EVENT MONITOR db2detaildeadlock
```

In a federated system environment in which an application accesses nicknames, the data requested by the application might not be available because of a deadlock at a data source. When this happens, DB2 relies on the deadlock handling facilities at the data source. If deadlocks occur across more than one data source, DB2 relies on data source timeout mechanisms to break the deadlock.

To log more information about deadlocks, set the database manager configuration parameter *diaglevel* to four. The logged information includes the locked object, the lock mode, and the application holding the lock. The current dynamic SQL and XQuery statements or static package names might also be logged. Dynamic SQL and XQuery statements are logged only at *diaglevel* four.

Related concepts:

- "DB2 architecture and process overview" on page 132
- "Lock granularity" on page 62

Lock waits and timeouts

Lock timeout detection is a database manager feature that prevents applications from waiting indefinitely for a lock to be released in an abnormal situation. For example, a transaction might be waiting for a lock held by another user's application, but the other user has left the workstation without allowing the application to commit the transaction that would release the lock. To avoid stalling an application in such a case, set the *locktimeout* configuration parameter to the maximum time that any application should wait to obtain a lock.

Setting this parameter helps avoid global deadlocks, especially in distributed unit of work (DUOW) applications. If the time that the lock request is pending is greater than the *locktimeout* value, the requesting application receives an error and its transaction is rolled back. For example, if *program1* tries to acquire a lock which is already held by *program2*, *program1* returns SQLCODE -911 (SQLSTATE 40001) with reason code 68 if the timeout period expires. The default value for *locktimeout* is -1, which turns off lock timeout detection.

Note: For table, row, data partition and MDC block locks, an application can override the database level *locktimeout* setting by using SET CURRENT LOCK TIMEOUT.

To log more information about lock-request timeouts in the db2diag.log, set the database manager configuration parameter *diaglevel* to four. The logged information includes the locked object, the lock mode, and the application holding the lock. The current dynamic SQL or XQuery statement or static package name might also be logged. A dynamic SQL or XQuery statement is logged only at *diaglevel* four.

You can get information about lock waits and lock timeouts from the lock wait information system monitor elements, or from the db.apps_waiting_locks health indicator.

Related concepts:

• "Specifying a lock wait mode strategy" on page 68

Related reference:

• "locktimeout - Lock timeout " on page 413

Specifying a lock wait mode strategy

An individual session can now specify a lock wait mode strategy, which is used when the session requires a lock that it cannot obtain immediately. The strategy indicates whether the session will:

- Return an SQLCODE and SQLSTATE when it cannot obtain a lock
- Wait indefinitely for a lock
- Wait a specified amount of time for a lock
- Use the value of the *locktimeout* database configuration parameter when waiting for a lock

The lock wait mode strategy is specified through the new SET CURRENT LOCK TIMEOUT statement, which changes the value of the CURRENT LOCK TIMEOUT special register. The CURRENT LOCK TIMEOUT special register specifies the number of seconds to wait for a lock before returning an error indicating that a lock cannot be obtained.

Traditional locking approaches can result in applications blocking each other. This happens when one application must wait for another application to release its lock. Strategies to deal with the impact of such blocking usually provide a mechanism to specify the maximum acceptable duration of the block. That is the amount of time that an application will wait prior to returning without a lock. Previously, this was only possible at the database level by changing the value of the *locktimeout* database configuration parameter.

Whereas the value of the *locktimeout* parameter applies to all locks, the lock types that are impacted by this new function include row, table, index key, and multidimensional clustering (MDC) block locks.

Related reference:

- "locktimeout - Lock timeout " on page 413

Preventing lock-related performance issues

Consider the following guidelines when you tune locking for concurrency and data integrity:

- Create small units of work with frequent COMMIT statements to promote concurrent access of data by many users.

  Include COMMIT statements when your application is logically consistent, that is, when the data you have changed is consistent. When a COMMIT is issued, locks are released except for table locks associated with cursors declared WITH HOLD.

- Specify an appropriate isolation level.

  Locks are acquired even if your application merely reads rows, so it is still important to commit read-only units of work. This is because shared locks are acquired by repeatable read, read stability, and cursor stability isolation levels in read-only applications. With repeatable read and read stability, all locks are held until a COMMIT is issued, preventing other processes from updating the locked data, unless you close your cursor using the WITH RELEASE clause. In addition, catalog locks are acquired even in uncommitted read applications using dynamic SQL or XQuery statements.

  The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but are not yet committed) unless you are using the uncommitted read isolation level.

- Use the LOCK TABLE statement appropriately.

  The statement locks an entire table. Only the table specified in the LOCK TABLE statement is locked. Parent and dependent tables of the specified table are not locked. You must determine whether locking other tables that can be accessed is necessary to achieve the desired result in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

  LOCK TABLE IN SHARE MODE

  You want to access data that is *consistent in time*; that is, data current for a table at a specific point in time. If the table experiences frequent activity, the only way to ensure that the entire table remains stable is to lock it. For example, your application wants to take a snapshot of a table. However, during the time your application needs to process some rows of a table, other applications are updating rows you have not yet processed. This is allowed with repeatable read, but this action is not what you want.

As an alternative, your application can issue the LOCK TABLE IN SHARE MODE statement: no rows can be changed, regardless of whether you have retrieved them or not. You can then retrieve as many rows as you need, knowing that the rows you have retrieved have not been changed just before you retrieved them.

With LOCK TABLE IN SHARE MODE, other users can retrieve data from the table, but they cannot update, delete, or insert rows into the table.

LOCK TABLE IN EXCLUSIVE MODE

You want to update a large part of the table. It is less expensive and more efficient to prevent all other users from accessing the table than it is to lock each row as it is updated, and then unlock the row later when all changes are committed.

With LOCK TABLE IN EXCLUSIVE MODE, all other users are locked out; no other applications can access the table unless they are uncommitted read applications.

- Use ALTER TABLE statements in applications.

  The ALTER TABLE statement with the LOCKSIZE parameter is an alternative to the LOCK TABLE statement. The LOCKSIZE parameter lets you specify a lock granularity of either ROW locks or TABLE locks for the next table access. For MDC tables, it also lets you specify a lock granularity of the BLOCKINSERT clause.

  The selection of ROW locks is no different from selecting the default lock size when a table is created. The selection of TABLE locks may improve query performance by limiting the number of locks that need to be acquired. However, concurrency might be reduced because all locks are on the complete table. For MDC tables, the selection of the BLOCKINSERT clause may improve the performance of INSERT operations by locking at the block level and avoiding row locks for insertions. Row-level locking is still performed for all other operations and is performed on key insertions to protect Repeatable Read (RR) scanners. The BLOCKINSERT option is useful for large insertions into cells by individual transactions. None of the LOCKSIZE choices prevent normal lock escalation.

- Close cursors to release the locks that they hold.

  When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, the database manager attempts to release all read locks that have been held for the cursor. Table read locks are IS, S, and U table locks. Row-read locks are S, NS, and U row locks. Block-read locks are IS, S, and U block locks.

  The WITH RELEASE clause has no effect on cursors that are operating under the CS or UR isolation levels. When specified for cursors that are operating under the RS or RR isolation levels, the WITH RELEASE clause ends some of the guarantees of those isolation levels. Specifically, a RS cursor may experience the *nonrepeatable read* phenomenon, and a RR cursor may experience either the *nonrepeatable read* or *phantom read* phenomenon.

  If a cursor that is originally RR or RS is reopened after being closed using the WITH RELEASE clause, then new read locks are acquired.

  In CLI applications, the DB2 CLI connection attribute SQL_ATTR_CLOSE_BEHAVIOR can be used to achieve the same results as CLOSE CURSOR WITH RELEASE.

- In a partitioned database environment, when you change the configuration parameters that affecting locking, ensure that the changes are made to all of the database partitions.

Related concepts:

- "Locks and concurrency control" on page 58
- "Lock attributes" on page 60
- "Lock granularity" on page 62

Lock type compatibility

Lock compatibility becomes an issue when one application currently has a lock on an object and another application requests a lock on the same object. When the two lock modes are compatible, the request for a second lock on the object can be granted.

If the lock mode of the requested lock is not compatible with the lock already held, the lock request cannot be granted. Instead, the request must wait until the first application releases its lock, and all other existing incompatible locks are released.

The following table displays information about the circumstances in which a lock request can be granted when another process holds or is requesting a lock on the same resource in a given state. A **no** indicates that the requestor must wait until all incompatible locks are released by other processes. Note that a timeout can occur when a requestor is waiting for a lock. A **yes** indicates that the lock is granted unless an earlier requestor is waiting for the resource.

Table 5. Lock Type Compatibility

	State of Held Resource											
State Being Requested	**none**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**X**	**Z**	**NW**	**W**
none	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
IS	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no	no
NS	yes	yes	yes	yes	yes	no	no	yes	no	no	yes	no
S	yes	yes	yes	yes	yes	no	no	yes	no	no	no	no
IX	yes	yes	yes	no	no	yes	no	no	no	no	no	no
SIX	yes	yes	yes	no	no	no	no	no	no	no	no	no
U	yes	yes	yes	yes	yes	no	no	no	no	no	no	no
X	yes	yes	no	no	no	no	no	no	no	no	no	no
Z	yes	no	no	no	no	no	no	no	no	no	no	no
NW	yes	yes	no	yes	no	no	no	no	no	no	no	yes
W	yes	yes	no	no	no	no	no	no	no	no	yes	no

Note:

I	Intent
N	None
NS	Next Key Share
S	Share
X	Exclusive
U	Update
Z	Super Exclusive
NW	Next Key Weak Exclusive
W	Weak Exclusive

Note:

- yes - **grant** lock requested immediately
- no - **wait** for held lock to be released or timeout to occur

Related concepts:

- "Lock attributes" on page 60
- "Locks and concurrency control" on page 58
- "Lock granularity" on page 62

Related reference:

- "Lock modes and access paths for standard tables" on page 74
- "Locking for block index scans for MDC tables" on page 85

Locks and data-access methods

An *access plan* is the method that the optimizer selects to retrieve data from a specific table. The access plan can have a significant effect on lock modes. For example, when an index scan is used to locate a specific row, the optimizer will probably choose row-level locking (IS) for the table. For example, if the EMPLOYEE table that has an index on employee number (EMPNO), access through an index might be used to select information for a single employee with a statement that contains the following SELECT clause:

```
SELECT *
  FROM EMPLOYEE
  WHERE EMPNO = '000310';
```

If an index is not used, the entire table must be scanned in sequence to find the selected rows, and may thus acquire a single table level lock (S). For example, if there is no index on the column SEX, a table scan might be used to select all male employees with a a statement that contains the following SELECT clause:

```
SELECT *
  FROM EMPLOYEE
  WHERE SEX = 'M';
```

Note: Cursor controlled processing uses the lock mode of the underlying cursor until the application finds a row to update or delete. For this type of processing, no matter what the lock mode of a cursor, an exclusive lock is always obtained to perform the update or delete.

Locking in range-clustered tables works slighly differently from standard key or next-key locking. In accessing a range of rows in a range-clustered table, all rows in the range are locked, even when some of those rows are empty. In standard key or next key locking, only rows with existing records are locked.

Reference tables provide detailed information about which locks are obtained for what kind of access plan.

Deferred access of the data pages implies that access to the row occurs in two steps, which results in more complex locking scenarios. The timing of lock aquisition and the persistence of the locks depend on the isolation level. Because the Repeatable Read isolation level retains all locks until the end of the transaction, the locks acquired in the first step are held and there is no need to acquire further locks in the second step. For the Read Stability and Cursor Stability isolation levels, locks must be acquired during the second step. To maximize concurrency, locks are not acquired during the first step and rely on the reapplication of all predicates to ensure that only qualifying rows are returned.

Related concepts:

- “Preventing lock-related performance issues” on page 69
- “Index types and next-key locking” on page 77
- “Lock attributes” on page 60
- “Locks and concurrency control” on page 58
- “Lock granularity” on page 62
- “Locks and types of application processing” on page 61

Related tasks:

- “Correcting lock escalation problems” on page 64

Related reference:

- “Lock modes and access paths for standard tables” on page 74
- “Lock modes for table and RID index scans of MDC tables” on page 81
- “Lock type compatibility” on page 71
- “Locking for block index scans for MDC tables” on page 85

Lock modes and access paths for standard tables

This topic includes reference information about locking methods for standard tables for different data-access plans.

The following tables list the types of locks obtained for standard tables at each level for different access plans. Each entry is made up of two parts: table lock and row lock. A dash indicates that a particular level of locking is not done.

Notes:

1. In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used.
2. Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 6. Lock Modes for Table Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 7. Lock Modes for Table Scans with Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Note: At UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks to an IS table lock and NS row locks.

Table 8. Lock Modes for RID Index Scans with no Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 9. Lock Modes for RID Index Scans with a Single Qualifying Row

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	IS/S	IX/U	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 10. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	IS/S	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 11. Lock Modes for RID Index Scans with Index and Other Predicates (sargs, resids) Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	IS/S	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

The following tables shows the lock modes for cases in which reading of the data pages is deferred to allow the list of rows to be:

- Further qualified using multiple indexes
- Sorted for efficient prefetching

Table 12. Lock modes for index scans used for deferred data page access: RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 13. Lock modes for index scans used for deferred data page access: after a RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 14. Lock modes for index scans used for deferred data page access: RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 15. Lock modes for index scans used for deferred data page access: RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		IX/X	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 16. Lock modes for index scans used for deferred data page access, after a RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IS/-	IX/U	IX/X	IX/U	IX/X

Table 17. Lock modes for index scans used for deferred data page access, after a RID index scan with predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Related concepts:

- "Lock attributes" on page 60
- "Lock granularity" on page 62

Related reference:

- "Lock modes for table and RID index scans of MDC tables" on page 81
- "Lock type compatibility" on page 71
- "Locking for block index scans for MDC tables" on page 85

Index types and next-key locking

As transactions cause changes to type-1 indexes, some next-key locking occurs. For type-2 indexes, minimal next-key locking occurs.

- Next-key locking for type 2 indexes

  Next-key locking occurs when a key is inserted into an index.

  During insertion of a key into an index, the row that corresponds to the key that will follow the new key in the index is locked only if that row is currently locked by an RR index scan. The lock mode used for the next-key lock is NW. This next-key lock is released before the key insertion is actually performed. Key insertion occurs when a row is inserted into a table.

  When updates to a row result in a change to the value of the index key for that row, key insertion also occurs because the original key value is marked deleted and the new key value is inserted into the index. For updates that affect only the include columns of an index, the key can be updated in place and no next-key locking occurs.

  During RR scans, the row that corresponds to the key that follows the end of the scan range is locked in S mode. If no keys follow the end of the scan range, an end-of-table lock is acquired to lock the end of the index. If the key that follows the end of the scan range is marked deleted, the scan continues to lock the corresponding rows until it finds a key that is not marked deleted, when it locks the corresponding row for that key, or until the end of the index is locked.

- Next-key locking for type-1 indexes:

  Next-key locks occur during deletes and inserts to indexes and during index scans. When a row is updated in, deleted from, or inserted into a table, an X lock is obtained on that row. For insertions this might be downgraded to a W lock.

  When the key is deleted from the table index or inserted into it, the table row that corresponds to the key that follows the deleted or inserted key in the index is locked. For updates that affect the value of the key, the original key value is

first deleted and the new value is inserted, so two next-key locks are acquired. The duration of these locks is determined as follows:

- During index key deletion, the lock mode on the next key is X and the lock is held until commit time.
- During index key insertion, the lock mode on the next key is NW. This lock is acquired only if there is contention for the lock, in which case the lock is released before the key is actually inserted into the index.
- During RR scans, the table row that corresponds to the key just beyond the end of the index scan range is locked in S mode and is held until commit time.
- During CS/RS scans, the row corresponding to the key just beyond the end of the index scan range is locked in NS mode if there is contention for the lock. This lock is released once the end of the scan range is verified.

The next-key locking for type-1 indexes during key insertions and key deletion might result in deadlocks. The following example shows how two transactions could deadlock. With type 2 indexes, such deadlocks do not occur.

Consider the following example of an index that contains 6 rows with the following values: `1 5 6 7 8 12`.

1. Transaction 1 deletes the row with key value 8. The row with value 8 is locked in X mode. When the corresponding key from the index is deleted, the row with value 12 is locked in X mode.
2. Transaction 2 deletes the row with key value 5. The row with value 5 is locked in X mode. When the corresponding key from the index is deleted, the row with value 6 is locked in X mode.
3. Transaction 1 inserts a row with key value 4. This row is locked in W mode. When inserting the new key into the index is attempted, the row with value 6 is locked in NW mode. This lock attempt will wait on the X lock that transaction 2 has on this row.
4. Transaction 2 inserts a row with key value 9. This row is locked in W mode. When inserting the new key into the index is attempted, the row with key value 12 is locked in NW mode. This lock attempt will wait on the X lock that transaction 1 has on this row.

When type-1 indexes are used, this scenario will result in a deadlock and one of these transactions will be rolled back.

Related concepts:

- "Index cleanup and maintenance" on page 340
- "Using relational indexes to improve performance" on page 25
- "Index reorganization" on page 318
- "Index structure" on page 26
- "Online index defragmentation" on page 341
- "Relational index performance tips" on page 30

Evaluate uncommitted data via lock deferral

To improve concurrency, DB2 now permits the deferral of row locks for CS or RS isolation scans in some situations until a record is known to satisfy the predicates of a query. By default, when row-locking is performed during a table or index scan, DB2 locks each row that is scanned before determining whether the row qualifies for the query. To improve the concurrency of scans, it may be possible to defer row locking until after it is determined that a row qualifies for a query.

To take advantage of this feature, enable the DB2_EVALUNCOMMITTED registry variable.

With this variable enabled, predicate evaluation can occur on uncommitted data. This means that a row that contains an uncommitted update may not satisfy the query, whereas if the predicate evaluation waited until the updated transaction completed, the row may satisfy the query. Additionally, uncommitted deleted rows are skipped during table scans. DB2 will skip deleted keys in type-2 index scans if the DB2_SKIPDELETED registry variable is enabled.

These registry variable settings apply at compile time for dynamic SQL or XQuery statements and at bind time for static SQL or XQuery statements. This means that even if the registry variable is enabled at runtime, the lock avoidance strategy is not employed unless DB2_EVALUNCOMMITTED was enabled at bind time. If the registry variable is enabled at bind time but not enabled at runtime, the lock avoidance strategy is still in effect. For static SQL or XQuery statements, if a package is rebound, the registry variable setting at bind time is the setting that applies. An implicit rebind of static SQL or XQuery statements will use the current setting of the DB2_EVALUNCOMMITTED.

Applicability of evaluate uncommitted for different access plans

Table 18. RID Index Only Access

Predicates	Evaluate Uncommitted
None	No
SARGable	Yes

Table 19. Data Only Access (relational or deferred RID list)

Predicates	Evaluate Uncommitted
None	No
SARGable	Yes

Table 20. RID Index + Data Access

Predicates		Evaluate Uncommitted	
Index	**Data**	**Index access**	**Data access**
None	None	No	No
None	SARGable	No	No
SARGable	None	Yes	No
SARGable	SARGable	Yes	No

Table 21. Block Index + Data Access

Predicates		Evaluate Uncommitted	
Index	**Data**	**Index access**	**Data access**
None	None	No	No
None	SARGable	No	Yes
SARGable	None	Yes	No
SARGable	SARGable	Yes	Yes

Example

The following example provides a comparison of the default locking behavior and the new evaluate uncommitted behavior.

The table below is the ORG table from the SAMPLE database.

```
DEPTNUMB DEPTNAME       MANAGER DIVISION   LOCATION
-------- -------------- ------- ---------- -------------
      10 Head Office        160 Corporate  New York
      15 New England         50 Eastern    Boston
      20 Mid Atlantic        10 Eastern    Washington
      38 South Atlantic      30 Eastern    Atlanta
      42 Great Lakes        100 Midwest    Chicago
      51 Plains             140 Midwest    Dallas
      66 Pacific            270 Western    San Francisco
      84 Mountain           290 Western    Denver
```

The following transactions are acting on this table, with the default Cursor Stability (CS) isolation level.

Table 22. Transactions on the ORG table with the CS isolation level

SESSION 1	SESSION 2
`connect to SAMPLE`	`connect to SAMPLE`
`+c update org set  deptnumb=5 where manager=160`	
	`select * from org where deptnumb >= 10`

The uncommitted UPDATE in Session 1 holds an exclusive record lock on the first row in the table, prohibiting the SELECT query in Session 2 from returning even though the row being updated in Session 1 does not currently satisfy the query in Session 2. This is because the CS isolation level dictates that any row accessed by a query must be locked while the cursor is positioned on that row. Session 2 cannot obtain a lock on the first row until Session 1 releases its lock.

When scanning the table, the lock-wait in Session 2 can be avoided using the evaluate uncommitted feature which first evaluates the predicate and then locks the row for a true predicate evaluation. As such, the query in Session 2 would not attempt to lock the first row in the table thereby increasing application concurrency. Note that this would also mean that predicate evaluation in Session 2 would occur with respect to the uncommitted value of deptnumb=5 in Session 1. The query in Session 2 would omit the first row in its result set despite the fact that a rollback of the update in Session 1 would satisfy the query in Session 2.

If the order of operations were reversed, concurrency could still be improved with evaluate uncommitted. Under default locking behavior, Session 2 would first acquire a row lock prohibiting the searched UPDATE in Session 1 from executing even though the UPDATE in Session 1 would not change the row locked by the query of Session 2. If the searched UPDATE in Session 1 first attempted to examine rows and then only lock them if they qualified, the query in Session 1 would be non-blocking.

Restrictions

The following external restrictions apply to this new functionality:

- The registry variable DB2_EVALUNCOMMITTED must be enabled.
- The isolation level must be CS or RS.
- Row locking is to occur.

- SARGable evaluation predicates exist.
- Evaluation uncommitted is not applicable to scans on the catalog tables.
- For MDC tables, block locking can be deferred for an index scan; however, block locking will not be deferred for table scans.
- Deferred locking will not occur on a table which is executing an inplace table reorg.
- Deferred locking will not occur for an index scan where the index is type-1.
- For Iscan-Fetch plans, row locking is not deferred to the data access but rather the row is locked during index access before moving to the row in the table.
- Deleted rows are unconditionally skipped for table scans while deleted type-2 index keys are only skipped if the registry variable DB2_SKIPDELETED is enabled.

Option to disregard uncommitted insertions

The DB2_SKIPINSERTED registry variable controls whether uncommitted insertions can be ignored for cursors using the Cursor Stability (CS) or Read Stability (RS) isolation levels. The DB2 database system can handle uncommitted insertions in the following ways:

- The DB2 database system can wait until the INSERT transaction completes (commits or rolls back) and process data accordingly. This is the default option, OFF.

  The following examples show instances when the default option, OFF, is preferred:

  – Suppose that two applications use a table to pass data between themselves with the first application inserting data into the table and the second one reading it. The data must be processed by the second application in the order presented in the table such that if the next row to be read is being inserted by the first application, the second application must wait until the insert is committed. In such cases, the default value for DB2_SKIPINSERTED should be used.

  – Suppose that an application modifies data by deleting the data and inserting the new image of the data. In such cases that avoid UPDATE statements, the default value for DB2_SKIPINSERTED should be used.
- The DB2 database system can ignore uncommitted insertions, which in many cases can improve concurrency. If you want this behavior, the registry variable must be specified as ON.

  In general, ON produces greater concurrency and is preferred for most applications. When the registry variable is enabled, uncommitted inserted rows are treated as if they had not yet been inserted.

Related concepts:

- "Evaluate uncommitted data via lock deferral" on page 78

Lock modes for table and RID index scans of MDC tables

In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used. The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not used.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 23. Lock Modes for Table Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 24. Lock Modes for Table Scans with Predicates on Dimension Columns Only

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

Table 25. Lock Modes for Table Scans with Other Predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

The following two tables show lock modes for RID indexes on MDC tables.

Table 26. Lock Modes for RID Index Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 27. Lock Modes for RID Index Scans with Single Qualifying Row

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 28. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 29. Lock Modes for RID Index Scans with Index Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 30. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Note: In the following tables, which shows lock modes for RID index scans used for deferred data-page access, at UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded to an IS table lock, an IS block lock, and NS row locks.

Table 31. Lock modes for RID index scans used for deferred data-page access: RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 32. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 33. Lock modes for RID index scans used for deferred data-page access: RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 34. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 35. Lock modes for RID index scans used for deferred data-page access: RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 36. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Related concepts:

- "Lock attributes" on page 60
- "Locks and concurrency control" on page 58
- "Lock granularity" on page 62

Related reference:

- "Lock modes and access paths for standard tables" on page 74
- "Locking for block index scans for MDC tables" on page 85
- "Lock type compatibility" on page 71

Locking for block index scans for MDC tables

The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not done.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 37. Lock Modes for Index Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 37. Lock Modes for Index Scans with No Predicates (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 38. Lock Modes for Index Scans with Dimension Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 39. Lock Modes for Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

Table 40. Lock Modes for Index Scans with Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

The following table lists lock modes for block index scans used for deferred data-page access:

Table 41. Lock modes for block index scans used for deferred data-page access: Block index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 42. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 43. Lock modes for block index scans used for deferred data-page access: Block index scan with dimension predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

Table 44. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with dimension predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

Table 45. Lock modes for block index scans used for deferred data-page access: Block index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 46. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

Table 47. Lock modes for block index scans used for deferred data-page access: Block index scan other predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 48. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with other predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Related concepts:

• "Lock granularity" on page 62

Related reference:

- "Lock modes for table and RID index scans of MDC tables" on page 81
- "Lock type compatibility" on page 71
- "Lock modes and access paths for standard tables" on page 74

Locking behavior on partitioned tables

In addition to an overall table lock, there is a lock for each data partition of a partitioned table. This allows for finer granularity and increased concurrency compared to a non-partitioned table. The new data partition lock is identified in the output of the db2pd command, event monitors, administrative views, and table functions.

When accessing a table, locking behavior obtains the table lock first, and then acquires data partition locks as dictated by the data accessed. Access methods and isolation levels might require locking of data partitions not in the result set. Once these data partition locks are acquired they might be held as long as the table lock. For example, a Cursor stability (CS) scan over an index might keep the locks on previously accessed data partitions to reduce the costs of re-acquiring the data partition lock if that data partition is referenced in subsequent keys. The data partition lock also carries the cost of ensuring access to the table spaces. For non-partitioned tables, table space access is handled by the table lock. Therefore, data partition locking occurs even if there is an exclusive or share lock at the table level for a partitioned table.

Finer granularity allows one transaction to have exclusive access to a given data partition and avoid row locking while other transactions are able to access other data partitions. This can be a result of the plan chosen for a mass update or due to escalation of locks to the data partition level. The table lock for many access methods is normally an intent lock, even if the data partitions are locked in share or exclusive. This allows for increased concurrency. However, if non-intent locks are required at the data partition level and the plan indicates that all data partitions may be accessed, then a non-intent might be chosen at the table level to prevent deadlocks between data partition locks from concurrent transactions.

Locking for SQL LOCK TABLE statements:

For partitioned tables, the only lock acquired for the LOCK TABLE statement is at the table level; no data partition locks are acquired. This ensures no row locking to the table for subsequent DML statements as well as avoiding deadlocks at the row, block, or data partition level. Using LOCK TABLE IN EXCLUSIVE MODE can also be used to guarantee exclusive access when updating indexes, which is useful in limiting the growth of type 2 indexes during a large update.

Effect of the LOCKSIZE TABLE parameter of the ALTER TABLE statement:

The ALTER TABLE statement has an option for setting LOCKSIZE TABLE, which ensures that the table is locked share or exclusive with no intent locks. For a partitioned table this lock strategy is applied to both the table lock and to the data partition locks for any data partitions accessed.

Row and block lock escalation:

For partitioned tables, the unit of escalation of row and block locks is to the data partition level. This again means that a table is more accessible to other

transactions even if a data partition is escalated to share, exclusive, or super exclusive, leaving other non-escalated data partitions are unaffected. The transaction might continue to row lock on other data partitions after escalation for a given data partition. The notification log message for escalations includes the data partition escalated as well as the partitioned table's name. Therefore, exclusive access to an index cannot be ensured by lock escalation. Either the statement must use an exclusive table level lock, an explicit LOCK TABLE IN EXCLUSIVE MODE statement must be issued, or the table must use the LOCKSIZE TABLE attribute. The overall table lock for the access method is chosen by the optimizer, and depends upon data partition elimination. A large update to a table might choose an exclusive table lock if there is no data partition elimination occurring.

Related concepts:

- "Lock attributes" on page 60
- "Lock granularity" on page 62
- "Locks and concurrency control" on page 58

Related reference:

- "Lock modes for table and RID index scans of MDC tables" on page 81
- "Lock type compatibility" on page 71
- "Locking for block index scans for MDC tables" on page 85
- "LOCK TABLE statement" in *SQL Reference, Volume 2*
- "LOCKS_HELD administrative view – Retrieve information on locks held" in *Administrative SQL Routines and Views*
- "LOCKWAITS administrative view – Retrieve current lockwaits information" in *Administrative SQL Routines and Views*
- "SNAPLOCK administrative view and SNAP_GET_LOCK table function – Retrieve lock logical data group snapshot information" in *Administrative SQL Routines and Views*
- "SNAPLOCKWAIT administrative view and SNAP_GET_LOCKWAIT table function – Retrieve lockwait logical data group snapshot information" in *Administrative SQL Routines and Views*
- "SNAPSHOT_LOCK table function" in *Administrative SQL Routines and Views*
- "SNAPSHOT_LOCKWAIT table function" in *Administrative SQL Routines and Views*

Chapter 6. The DB2 Governor

The Governor utility

The governor can monitor the behavior of applications that run against a database and can change certain behavior, depending on the rules that you specify in the governor configuration file.

A governor instance consists of a front-end utility and one or more daemons. Each instance of the governor that you start is specific to an instance of the database manager. By default, when you start the governor a governor daemon starts on each database partition of a partitioned database. However, you can specify that a daemon be started on a single database partition that you want to monitor.

Note: When the governor is active, its snapshot requests might affect database manager performance. To improve performance, increase the governor wake-up interval to reduce its CPU usage.

Each governor daemon collects information about the applications that run against the database. It then checks this information against the rules that you specify in the governor configuration file for this database.

The governor manages application transactions as specified by the rules in the configuration file. For example, applying a rule might indicate that an application is using too much of a particular resource. The rule would specify the action to take, such as to change the priority of the application or force it to disconnect from the database.

If the action associated with a rule changes the priority of the application, the governor changes the priority of agents on the database partition where the resource violation occurred. In a partitioned database, if the application is forced to disconnect from the database, the action occurs even if the daemon that detected the violation is running on the coordinator node of the application.

The governor logs any actions that it takes. To review the actions, you query the log files.

Related concepts:

- "Governor log files" on page 102
- "The governor configuration file" on page 95
- "The Governor daemon" on page 92

Related tasks:

- "Configuring the Governor" on page 94
- "Starting and stopping the governor" on page 93

Related reference:

- "db2gov - DB2 governor command" in *Command Reference*

The Governor daemon

When the governor daemon starts, either when you execute by the `db2gov` utility or when it wakes up, it runs the following task loop.

1. It checks whether its governor configuration file has changed or has not yet been read. If either condition is true, the daemon reads the rules in the file. This allows you to change the behavior of the governor daemon while it is running.
2. It requests snapshot information about resource-use statistics for each application and agent that is working on the database.

   Note: On some platforms, the CPU statistics are not available from the DB2 Monitor. In this case, the account rule and the CPU limit are not available.
3. It checks the statistics for each application against the rules in the governor configuration file. If a rule applies to an application, the governor performs the specified action.

   Note: The governor compares accumulated information with the values defined in the configuration file. This means that if the configuration file is updated with new values that an application may have already breached, the governor rules concerning that breach are applied immediately to the application at the next governor interval.
4. It writes a record in the governor log file for any action that it takes.

Note: The governor cannot be used to adjust agent priorities if the *agentpri* database manager configuration parameter is anything other than the system default.

When the governor finishes its tasks, it sleeps for the interval specified in the configuration file. when the interval elapses, the governor wakes up and begins the task loop again.

When the governor encounters an error or stop signal, it does cleanup processing before it ends. Using a list of applications whose priorities have been set, the cleanup processing resets all application agent priorities. It then resets the priorities of any agents that are no longer working on an application. This ensures that agents do not remain running with nondefault priorities after the governor ends. If an error occurs, the governor writes a message to the administration notification log to indicate that it ended abnormally.

Note: Although the governor daemon is not a database application, and therefore does not maintain a connection to the database, it does have an instance attachment. Because it can issue snapshot requests, the governor daemon can detect when the database manager ends.

Related concepts:

- "The Governor utility" on page 91

Related tasks:

- "Starting and stopping the governor" on page 93

Starting and stopping the governor

The governor utility monitors applications that connect to a database and changes their behavior according to rules that you specify in a governor configuration file for that database.

Prerequisites:

Before you start the governor, you must create the configuration file.

Restrictions:

To start or stop the governor, you must have *sysadm* or *sysctrl* authorization.

Procedure:

To start or stop the governor:

1. To start the governor, execute the *db2gov* command at the DB2 command line. Enter the following required parameters:
   - START *database_name*

     The database name that you specify must match the name of the database in the configuration file that you specify. An error is returned if the names are not the same. Note that if a governor is running for more than one database, daemons will be started for each database.
   - *config_file_name*

     The name of the configuration file for the governor on this database. If the file is not in the default location, which is the `sqllib` directory, you must include the path as well as the file name.
   - *log_file_name*

     The base name of the log file for this governor. On a partitioned database, the database partition number is added for each database partition where a daemon runs for this instance of the governor.

   To start the governor on a single database partition for a partitioned database, add the *nodenum* option. For example, to start the governor for a database called *sales* on only node 3 of a partitioned database with a configuration file called *salescfg* and a log file called *saleslog*, enter the following command:

   ```
   db2gov START sales nodenum 3 salescfg saleslog
   ```

   To start the governor on all database partitions of the *sales* database, enter the following command:

   ```
   db2gov START sales  salescfg saleslog
   ```
2. To stop the governor, enter the *db2gov* command with the STOP option.

   For example, to stop the governor on all database partitions of the *sales* database, enter the following command:

   ```
   db2gov STOP sales
   ```

   To stop the governor on only database partition 3, enter the following command:

   ```
   db2gov START sales nodenum 3
   ```

Related concepts:

- "The Governor daemon" on page 92
- "The Governor utility" on page 91

Related reference:

- "db2gov - DB2 governor command" in *Command Reference*

Configuring the Governor

To configure the Governor, you create a configuration file that determines the database that an instance of the Governor monitors and how it manages queries.

The configuration file consists of a set of rules. The first three rules specify the database to monitor, the interval at which to write log records, and the interval at which to wake up for monitoring. The remaining rules specify how to monitor the database server and what actions to take in specific circumstances.

Procedure:

To create a Governor configuration file:

1. In a directory that is mounted or available from all database partitions, create an ASCII file with a descriptive name. For example, the configuration file for a governor instance that monitors the **sales** database might be called *govcfgsales*.
2. Open the file in any text editor and enter configuration information and action conditions.

   End each rule with a semicolon (;). The following configuration information is recommended:

   - dbname: The name or alias of the database to be monitored.
   - account: The number of minutes after which the governor instance writes CPU usage statistics to its log file.
   - interval: The number of seconds after which the governor daemon wakes up to monitor activity. If you do not specify an interval, the default value of 120 seconds is used.

   For example, the first three rules in the configuration file might look like this:

   ```
   { Wake up once a second, the database name is sales,
    do accounting every 30 minutes.  }
   interval 1; dbname sales; account 30;
   ```

   Add rules that specify the conditions to monitor and the action to take if the rule evaluates to true. For example, you might add a rule that limits to an hour the amount of time that a unit of work (UOW) can run before being forced to disconnect from the database, as follows:

   ```
   setlimit uowtime 3600 action force;
   ```

3. Save the file.

Related concepts:

- "Example of a Governor configuration file" on page 101
- "Governor log files" on page 102
- "Governor rule elements" on page 97
- "The governor configuration file" on page 95
- "System monitor switches" in *System Monitor Guide and Reference*

Related reference:

- "appl_priority - Application Agent Priority monitor element" in *System Monitor Guide and Reference*
- "db2gov - DB2 governor command" in *Command Reference*

The governor configuration file

When you start the governor, you specify the configuration file that contains the rules that govern applications running against the database. The governor evaluates each rule and acts as specified when the rule evaluates to true.

If your rule requirements change, you edit the configuration file without stopping the governor. Each governor daemon detects that the file has changed, and rereads it.

The configuration file must be created in a directory that is mounted across all the database partitions so that the governor daemon on each database partition can read the same configuration file.

The configuration file consists of three required rules that identify the database to be monitored, the interval at which log records are written, and the sleep interval of the governor daemons. Following these parameters, the configuration file contains a set of optional application-monitoring rules and actions. The following comments apply to all rules:

- Delimit comments inside { } braces.
- Most entries can be specified in uppercase, lowercase, or mixed case characters. The exception is the application name, specified as an argument to the `applname` rule, which is case sensitive.
- Each rule ends with a semicolon (;).

Required rules

The following rules specify the database to be monitored and the interval at which the daemon wakes up after each loop of activities. Each of these rules is specified only once in the file.

dbname
The name or alias of the database to be monitored.

account *nnn*
Account records are written containing CPU usage statistics for each connection at the specified number of minutes.

Note: This option is not available in the Windows® environment.

If a short connect session occurs entirely within the account interval, no log record is written. When log records are written, they contain CPU statistics that reflect CPU usage since the previous log record for the connection. If the governor is stopped then restarted, CPU usage may be reflected in two log records; these can be identified through the application IDs in the log records.

interval
The interval, in seconds, at which the daemon wakes up. If you do not specify an interval, the default value, 120 seconds, is used.

Rules that govern actions

Following the required rules, you can add rules that specify how to govern the applications. These rules are made of smaller components called rule clauses. If used, the clauses must be entered in a specific order in the rule statement, as follows:

1. **desc** (optional): a comment about the rule, enclosed in quotation marks
2. **time** (optional): the time during the day when the rule is evaluated
3. **authid** (optional): one or more authorization IDs under which the application executes statements
4. **applname** (optional): the name of the executable or object file that connects to the database. This name is case sensitive. The application name must be surrounded by double quotes if the application contains spaces.
5. **setlimit**: the limits that the governor checks. These can be one of several, for example, CPU time, number of rows returned, or idle time..
6. **action** (optional): the action to take if a limit is reached. If no action is specified, the governor reduces the priority of agents working for the application by 10 when a limit is reached. Actions against the application can include reducing its agent priority, forcing it to disconnect from the database, or setting scheduling options for its operations.

You combine the rule clauses to form a rule, using each clause only once in each rule, and end the rule with a semicolon, as shown in the following examples:

```
desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;
```

If more than one rule applies to an application, all are applied. Usually, the action associated with the rule limit encountered first is the action that is applied first. An exception occurs you specify if -1 for a clause in a rule. In this case, the value specified for the clause in the subsequent rule can only override the value previously specified for the *same* clause: other clauses in the previous rule are still operative. For example, one rule uses the `rowssel 100000 uowtime 3600` clauses to specify that the priority of an application is decreased either if its elapsed time is greater than 1 hour or if it selects more than 100 000 rows. A subsequent rule uses the `uowtime -1` clause to specify that the same application can have unlimited elapsed time. In this case, if the application runs for more than 1 hour, its priority is not changed. That is, `uowtime -1` overrides `uowtime 3600`. However, if it selects more than 100 000 rows, its priority is lowered because `rowssel 100000` is still valid.

Order of rule application

The governor processes rules in the configuration file from the top of the file to the bottom. However, if a later rule's **setlimit** clause is more relaxed than a preceding rule, the more restrictive rule still applies. For example, in the following configuration file, `admin` will be limited to 5000 rows despite the later rule because the first rule is more restrictive.

```
desc "Force anyone selecting 5000 or more rows"
setlimit rowssel 5000 action force;

desc "Allow user admin to select more rows"
authid admin
setlimit rowssel 10000 action force;
```

To ensure that a less restrictive rule overrides a more restrictive rule that occurs earlier in the file, you can specify the -1 option to clear the previous rule before applying the new one. For example, in the following configuration file, the initial

rule limits all users to 5000 rows. The second rule clears this limit for admin, and the third rule resets the limit for admin to 10000 rows.

```
desc "Force anyone selecting 5000 or more rows"
setlimit rowssel 5000 action force;

desc "Clear the rowssel limit for admin"
authid admin
setlimit rowssel -1;

desc "Now set the higher rowssel limit for admin"
authid admin
setlimit rowssel 10000 action force;
```

Related concepts:

- "Example of a Governor configuration file" on page 101
- "Governor rule elements" on page 97

Governor rule elements

Each rule in the governor configuration file is made up of clauses that specify the conditions for applying the rule and the action that results if the rule evaluates to true. The clauses must be specified in the order shown. In the clause descriptions, [ ] indicates an optional clause.

Optional beginning elements

[desc] Specifies a text description for the rule. The description must be enclosed by either single or double quotation marks.

[time] Specifies the time period during which the rule is to be evaluated.

The time period must be specified in the following format `time hh:mm hh:mm`, for example, `time 8:00 18:00`. If this clause is not specified, the rule is valid 24 hours a day.

[authid]

Specifies one or more authorization IDs (authid) under which the application is executing. Multiple authids must be separated by a comma (,), for example `authid gene, michael, james`. If this clause does not appear in a rule, the rule applies to all authids.

[applname]

Specifies the name of the executable (or object file) that makes the connection to the database.

Multiple application names must be separated by a comma (,), for example, `applname db2bp, batch, geneprog`. If this clause does not appear in a rule, the rule applies to all application names.

Notes:

1. Application names are case sensitive.
2. The database manager truncates all application names to 20 characters. You should ensure that the application you want to govern is uniquely identified by the first 20 characters of its application name; otherwise, an unintended application may be governed.

   Application names specified in the governor configuration file are truncated to 20 characters to match their internal representation.

Limit clauses

setlimit

Specifies one or more limits for the governor to check. The limits can only be -1 or greater than 0 (for example, `cpu -1 locks 1000 rowssel 10000`). At least one of the limits (cpu, locks, rowsread, uowtime) must be specified, and any limit not specified by the rule is not limited by that particular rule. The governor can check the following limits:

cpu *nnn*

Specifies the number of CPU seconds that can be consumed by an application. If you specify `-1`, the governor does not limit the application's CPU usage.

locks *nnn*

Specifies the number of locks that an application can hold. If you specify `-1`, the governor does not limit the number of locks held by the application.

rowssel *nnn*

Specifies the number of rows that are returned to the application. This value will only be non-zero at the coordinator node. If you specify `-1`, the governor does not limit the number of rows that can be selected. The maximum value that can be specified for *nnn* is 4 294 967 298.

uowtime *nnn*

Specifies the number of seconds that can elapse from the time that a unit of work (UOW) first becomes active. If you specify `-1`, the elapsed time is not limited.

Note: If you used the `sqlmon` (Database System Monitor Switch) API to deactivate the unit of work monitor switch or the timestamp monitor switch, this will affect the ability of the governor to govern applications based on the unit of work elapsed time. The governor uses the monitor to collect information about the system. If you turn off the switches in the database manager configuration file, then it is turned off for the entire instance, and governor will no longer receive this information.

idle *nnn*

Specifies the number of idle seconds allowed for a connection before a specified action is taken. If you specify `-1`, the connection's idle time is not limited.

rowsread *nnn*

Specifies the number of rows an application can select. If you specify `-1`, there is no limit on the number of rows the application can select. The maximum value that can be specified for *nnn* is 4 294 967 298.

Note: This limit is not the same as rowssel. The difference is that rowsread is the count of the number of rows that had to be read in order to return the result set. The number of rows read includes reads of the catalog tables by the engine and may be diminished when indices are used.

Action clauses

[action]

Specifies the action to take if one or more of the specified limits is exceeded. You can specify the following actions.

Note: If a limit is exceeded and the action clause is not specified, the governor reduces the priority of agents working for the application by 10.

nice *nnn*

Specifies a change to the relative priority of agents working for the application. Valid values are from –20 to +20.

For this parameter to be effective:

- On UNIX® -based platforms, the *agentpri* database manager parameter must be set to the default value; otherwise, it overrides the priority clause.
- On Windows platforms, the *agentpri* database manager parameter and *priority* action may be used together.

force Specifies to force the agent that is servicing the application. (Issues a FORCE APPLICATION to terminate the coordinator agent.)

Note: In multi-partition database environments, the force action is only carried out when the governor daemon is running on the application's coordinating database partition. Therefore if a governor daemon is running on database partition A and a limit is exceeded for some application whose coordinating database partition is database partition B, then the force action is skipped.

schedule [class]

Scheduling improves the priorities of the agents working on the applications with the goal of minimizing the average response times while maintaining fairness across all applications.

The governor chooses the top applications for scheduling based on the following three criteria:

- The application holding the most locks

  This choice is an attempt to reduce the number of lockwaits.
- The oldest application
- The application with the shortest estimated remaining running time.

  This choice is an attempt to allow as many short-lived statements as possible to complete during the interval.

The top three applications in each criteria are given higher priorities than all other applications That is, the top application in each criterion group is given the highest priority, the next highest applications are given the second highest priority and the third-ranked applications are given the third highest priority. If a single application is ranked in the top three in more than one of the criteria, it is given the appropriate priority for the criterion in which it ranked highest, and the next highest application is given the next highest priority for the other criteria. For example, if application A holds the most locks but has the third shortest estimated remaining running time, it is given the highest priority

for the first criterion, and the fourth ranked application with the shortest estimated remaining running time is given the third highest priority for that criterion.

The applications selected by this governor rule are divided in up to three classes. For each class, the governor chooses nine applications, which are the top three applications from each class, based on the criteria listed above. If you specify the class option, all applications selected by this rule are considered a single class, and nine applications are chosen and given higher priorities as described above.

If an application is selected in more than one governor rule, it is governed by the last rule in which is it selected.

Note: If you used the `sqlmon` (Database System Monitor Switch) API to deactivate the statement switch, this will affect the ability of the governor to govern applications based on the statement elapsed time. The governor uses the monitor to collect information about the system. If you turn off the switches in the database manager configuration file, then it is turned off for the entire instance, and governor will no longer receive this information.

The schedule action can:

- Ensure that applications in different groups each get time without all applications splitting time evenly.

  For instance, if 14 applications (three short, five medium, and six long) are running at the same time, they may all have poor response times because they are splitting the CPU. The database administrator can set up two groups, medium-length applications and long-length applications. Using priorities, the governor permits all the short applications to run, and ensures that at most three medium and three long applications run simultaneously. To achieve this, the governor configuration file contains one rule for medium-length applications, and another rule for long applications.

  The following example shows a portion of a governor configuration file that illustrates this point:

  ```
  desc "Group together medium applications in 1 schedule class"
  applname medq1, medq2, medq3, medq4, medq5
  setlimit cpu -1
  action schedule class;

  desc "Group together long applications in 1 schedule class"
  applname longq1, longq2, longq3, longq4, longq5, longq6
  setlimit cpu -1
  action schedule class;
  ```

- Ensure that each of several user groups (for example, organizational departments) gets equal prioritization.

  If one group is running a large number of applications, the administrator can ensure that other groups are still able to obtain reasonable response times for their applications. For instance, in a case involving three departments (Finance, Inventory, and Planning), all the Finance users could be put into one group, all the Inventory users could be put into a second, and all the Planning users could be put into a third group. The processing

power would be split more or less evenly among the three departments. The following example shows a portion of a governor configuration file that illustrates this point:

```
desc "Group together Finance department users"
authid tom, dick, harry, mo, larry, curly
setlimit cpu -1
action schedule class;

desc "Group together Inventory department users"
authid pat, chris, jack, jill
setlimit cpu -1
action schedule class;

desc "Group together Planning department users"
authid tara, dianne, henrietta, maureen, linda, candy
setlimit cpu -1
action schedule class;
```

- Let the governor schedule all applications.

  If the class option is not included with the action, the governor creates its own classes based on how many active applications fall under the schedule action, and puts applications into different classes based on the DB2 query compiler's cost estimate for the query the application is running. The administrator can choose to have all applications scheduled by not qualifying which applications are chosen. That is, no *applname* or *authid* clauses are supplied, and the *setlimit* clause causes no restrictions.

Note: If a limit is exceeded and the action clause is not specified, the governor reduces the priority of agents working for the application.

Related concepts:

- "Example of a Governor configuration file" on page 101
- "The governor configuration file" on page 95

Related tasks:

- "Configuring the Governor" on page 94

Example of a Governor configuration file

The following example shows a governor configuration file that sets several rules with actions:

```
{ Wake up once a second, the database name is ibmsampl,
 do accounting every 30 minutes.  }
interval 1; dbname ibmsampl; account 30;

desc "CPU restrictions apply 24 hours a day to everyone"
setlimit cpu 600 rowssel 1000000 rowsread 5000000;

desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc 'Slow down a subset of applications'
applname jointA, jointB, jointC, quryA
setlimit cpu 3 locks 1000 rowssel 500 rowsread 5000;

desc "Have governor prioritize these 6 long apps in 1 class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
```

```
action schedule class;

desc "Schedule all applications run by the planning dept"
authid planid1, planid2, planid3, planid4, planid5
setlimit cpu -1
action schedule;

desc "Schedule all CPU hogs in one class which will control consumption"
setlimit cpu 3600
action schedule class;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;

desc "During day hours do not let anyone run for more than 10 seconds"
time 8:30 17:00 setlimit cpu 10 action force;

desc "Allow users doing performance tuning to run some of
     their applications during lunch hour"
time 12:00 13:00 authid ming, geoffrey, john, bill
applname tpcc1, tpcc2, tpcA, tpvG setlimit cpu 600 rowssel 120000 action force;

desc "Some people should not be limited -- database administrator
  and a few others.  As this is the last specification in the
  file, it will override what came before."
authid gene, hershel, janet setlimit cpu -1 locks -1 rowssel -1 uowtime -1;

desc "Increase the priority of an important application so it always
      completes quickly"
applname V1app setlimit cpu 1 locks 1 rowssel 1 action priority -20;
```

Related concepts:

- "Governor rule elements" on page 97
- "The governor configuration file" on page 95

Related tasks:

- "Configuring the Governor" on page 94

Governor log files

Whenever a governor daemon performs an action, it writes a record to its log file. Actions include the following:

- Forcing an application
- Reading the governor configuration file
- Changing an application priority
- Encountering an error or warning
- Starting or ending

Each governor daemon has a separate log file. Separate log files prevent file-locking bottlenecks that might result when many governor daemons write to the same file at the same time. To merge the log files together and query them, use the `db2govlg` utility.

The log files are stored in the `log` subdirectory of the `sqllib` directory, except on Windows Operating systems, where the `log` subdirectory is under the instance directory. You provide the base name for the log file when you start the governor with the `db2gov` command. Make sure that the log file name contains the database

name to distinguish log files on each database partition that is governed. To ensure that the filename is unique for each governor in a partitioned database environment, the database partition number where the governor daemon runs is automatically appended to the log file name.

Log file record format

Each record in the log file has the following format:

```
Date Time NodeNum RecType Message
```

Note: The format of the *Date* and *Time* fields is yyyy-mm-dd hh.mm.ss. You can merge the log files for each database partition by sorting on this field.

The *NodeNum* field indicates the number of the database partition on which the governor is running.

The *RecType* field contains different values, depending on the type of log record being written to the log. The values that can be recorded are:

- `START`: the governor was started
- `STOP`: the governor was stopped
- `FORCE`: an application was forced
- `NICE`: the priority of an application was changed
- `ERROR`: an error occurred
- `WARNING`: a warning occurred
- `READCFG`: the governor read the configuration file
- `ACCOUNT`: the application accounting statistics.
- `SCHEDGRP`: a change in agent priorities occurred.

Some of these values are described in more detail below.

START
The START record is written when the governor is started. It has the following format:

```
Database = <database_name>
```

STOP The STOP record is written when the governor is stopped. It has the following format:

```
Database = <database_name>
```

FORCE
The FORCE record is written out whenever the governor determines that an application is to be forced as required by a rule in the governor configuration file. The FORCE record has the following format:

```
<appl_name> <auth_id> <appl_id> <coord_partition> <cfg_line>
<restriction_exceeded>
```

where:

<coord_partition>
Specifies the number of the application's coordinating database partition.

<cfg_line>
Specifies the line number in the governor configuration file where the rule causing the application to be forced is located.

<restriction_exceeded>
Provides details about how the rule was exceeded. This can contain the following values:

- `CPU`: the total application USR cpu plus SYS cpu time in seconds
- `Locks`: the total locks held by the application
- `Rowssel`: the total rows selected by the application
- `Rowsread`: the total rows read by the application
- `Idle`: amount of time the application was idle
- `ET (elapsed time)`: elapsed time since the application's current unit of work started (the uowtime setlimit was exceeded)

NICE The NICE record is written when the priority of an application is changed via a priority action specified in the governor configuration file. The NICE record has the following format:

```
<appl_name> <auth_id> <appl_id> <nice value> (<cfg_line>)
<restriction_exceeded>
```

where:

<nice value>
Specifies the increment (or decrement) that will be made to the priority value for the application's agent process.

<cfg_line>
Specifies the line number in the governor configuration file where the rule causing the application's priority to be changed is located.

<restriction_exceeded>
Provides details about how the rule was exceeded. This can contain the following values:

- `CPU`: the total application USR cpu plus SYS cpu time in seconds
- `Locks`: the total locks held by the application
- `Rowssel`: the total rows selected by the application
- `Rowsread`: the total rows read by the application
- `Idle`: amount of time the application was idle
- `ET (elapsed time)`: elapsed time since the application's current unit of work started (the uowtime setlimit was exceeded)

ERROR
The ERROR record is written when the governor daemon needs to shut down.

WARNING
The WARNING record is written to the governor log in the following situations:

- The sqlefrce API was called to force an application, but it returned a positive SQLCODE.
- A snapshot call returned a positive SQLCODE that was not 1611 (″SQL1661 No data was returned″).
- A snapshot call returned a negative SQLCODE that was not −1224 (″SQL 1224N A database agent could not be started to service a request, or was terminated as a result of a database system shutdown or a force command″) or −1032 (″SQL1032N No start database manager command was issued″). These return codes occur when a previously active instance has been brought down.

- In a UNIX environment, an attempt was made to install a signal handler and the attempt failed.

ACCOUNT

The ACCOUNT record is written to the governor log in the following situations:

- The value of agent_usr_cpu or agent_sys_cpu for this application has changed since the last ACCOUNT record was written for this application.
- An application is found to be no longer active

The ACCOUNT record has the following format:

<auth_id> <appl_id> <applname> <connect time> <agent_usr_cpu delta> <agent_sys_cpu delta>

SCHEDGRP

The SCHEDGRP record is written whenever:

- An application is added to a scheduling group
- An application is moved from one scheduling group to another.

The SCHEDGRP record has the following format:

```
<appl_name> <auth_id> <appl_id> <cfg_line> <restriction_exceeded>
```

where:

<cfg_line>
Specifies the line number in the governor configuration file where the rule causing the application to be scheduled is located.

<restriction_exceeded>
Provides details about how the rule was exceeded. This can contain the following values:

- `CPU`: the total application USR cpu plus SYS cpu time in seconds
- `Locks`: the total locks held by the application
- `Rowssel`: the total rows selected by the application
- `Rowsread`: the total rows read by the application
- `Idle`: amount of time the application was idle
- `ET (elapsed time)`: elapsed time since the application's current unit of work started (the uowtime setlimit was exceeded)

Because standard values are written, you can query the log files for different types of actions. The *Message* field provides other nonstandard information that varies according to the value under the *RecType* field. For instance, a `FORCE` or `NICE` record indicates application information in the *Message* field, while an `ERROR` record includes an error message.

An example log file is as follows:

```
1995-12-11 14.54.52    0 START      Database = TQTEST
1995-12-11 14.54.52    0 READCFG    Config = /u/db2instance/sqllib/tqtest.cfg
1995-12-11 14.54.53    0 ERROR      SQLMON Error: SQLCode = -1032
1995-12-11 14.54.54    0 ERROR      SQLMONSZ Error: SQLCode = -1032
```

Related concepts:

- "Governor log file queries" on page 106
- "The Governor utility" on page 91

Governor log file queries

Each governor daemon writes to its own log file. You can use `db2govlg` utility to query the log file. You can list the log files for a single database partition, or for all database partitions, sorted by date and time. You can also query on the basis of the *RecType* log field. The syntax for `db2govlg` is as follows:

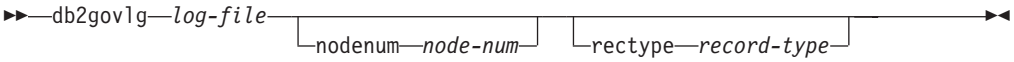


Figure 11. Syntax for db2govlg

The parameters are as follows:

log-file
The base name of the log file (or files) that you want to query.

nodenum **node-num**
The node number of the database partition on which the governor is running.

rectype **record-type**
The type of record that you want to query. The record types are:
- START
- READCFG
- STOP
- FORCE
- NICE
- ERROR
- WARNING
- ACCOUNT

There are no authorization restrictions for using this utility. This allows all users to query whether the governor has affected their application. If you want to restrict access to this utility, you can change the group permissions for the `db2govlg` file.

Related concepts:

- "Governor log files" on page 102
- "The Governor utility" on page 91

Related reference:

- "db2govlg - DB2 governor log query command" in *Command Reference*

Chapter 7. Maximizing I/O Efficiency

Reducing I/O by Tuning SQL and XQuery Statements

Specifying row blocking to reduce overhead

Row blocking reduces database manager overhead for cursors by retrieving a *block* of rows in a single operation.

Note: The block of rows that you specify is a number of pages in memory. It is not a multi-dimensional (MDC) table block, which is physically mapped to an extent on disk.

Row blocking levels are specified by the following arguments to the BIND or PREP commands:

UNAMBIG
Blocking occurs for read-only cursors and cursors not specified as "FOR UPDATE OF". Ambiguous cursors are treated as updateable.

ALL Blocking occurs for read-only cursors and cursors not specified as "FOR UPDATE OF". Ambiguous cursors are treated as read-only.

NO Blocking does not occur for any cursors. Ambiguous cursors are treated as read-only.

Prerequisites:

Two database manager configuration parameters must be set appropriately. Both values are set as a number of pages of memory. Note the values of these parameters for use in block-size calculations.

- The database manager configuration parameter *aslheapsz* specifies application support layer heap size for local applications.
- The database manager configuration parameter *rqrioblk* specifies the size of the communication buffer between remote applications and their database agents on the database server.

Procedure:

To specify row blocking:

1. Use the values of the *aslheapsz* and *rqrioblk* configuration parameters to estimate how many rows are returned for each block. In both formulas *orl* is the output row length in bytes.
   - Use the following formula for local applications:

     ```
     Rows per block = aslheapsz * 4096 / orl
     ```

     The number of bytes per page is 4 096.
   - Use the following formula for remote applications:

     ```
     Rows per block = rqrioblk / orl
     ```
2. To enable row blocking, specify an appropriate argument to the BLOCKING option in the PREP or BIND commands.

If you do not specify a BLOCKING option, the default row blocking type is UNAMBIG. For the command line processor and call level interface, the default row blocking type is ALL.

Note: If you use the FETCH FIRST *n* ROWS ONLY clause or the OPTIMIZE FOR *n* ROWS clause in a SELECT statement, the number of rows per block will be the minimum of the following:
- The value calculated in the above formula
- The value of *n* in the FETCH FIRST clause
- The value of *n* in the OPTIMIZE FOR clause

Related reference:
- "aslheapsz - Application support layer heap size " on page 403
- "rqrioblk - Client I/O block size " on page 406

Efficient SELECT statements

Because SQL is a flexible high-level language, you can write several different SELECT statements to retrieve the same data. However, the performance might vary for the different forms of the statement as well as for the different classes of optimization.

Consider the following guidelines for SELECT statements:
- Specify only columns that you need. Although it is simpler to specify all columns with an asterisk (*), unnecessary processing and return of unneeded columns results.
- Use predicates that restrict the answer set to only those rows that you require
- When the number of rows you need is significantly less than the total number of rows that might be returned, specify the OPTIMIZE FOR clause. This clause affects both the choice of access plans and the number of rows that are blocked in the communication buffer.
- When the number of rows to be retrieved is small, specify only the OPTIMIZE FOR *k* ROWS clause. You do not need the FETCH FIRST n ROWS ONLY clause. However, if *n* is large and you want the first *k* rows quickly with a possible delay for the subsequent *k* rows, specify both clauses. The size of the communication buffers is based on the lesser of *n* and *k*. The following example shows both clauses:
```
SELECT EMPNAME, SALARY FROM EMPLOYEE
   ORDER BY SALARY DESC
   FETCH FIRST 100 ROWS ONLY
   OPTIMIZE FOR 20 ROWS
```
- To take advantage of row blocking, specify the FOR READ ONLY or FOR FETCH ONLY clause to improve performance. In addition, concurrency improves because exclusive locks are never held on the rows retrieved. Additional query rewrites can also occur. Specifying the FOR READ ONLY or FOR FETCH ONLY clause as well as the BLOCKING ALL BIND option can improve the performance of queries against nicknames in a federated system in a similar way.
- For cursors that will be updated with positioned updates, specify the FOR UPDATE OF clause to allow the database manager to choose more appropriate locking levels initially and avoid potential deadlocks. Note that FOR UPDATE cursors cannot take advantage of row blocking.

- For cursors that will be updated with searched updates, you can avoid deadlocks and still allow row blocking by forcing U locks on affected rows with the FOR READ ONLY and the USE AND KEEP UPDATE LOCKS clauses.
- Avoid numeric data type conversions whenever possible. When comparing values, it might be more efficient to use items that have the same data type. If conversions are necessary, inaccuracies due to limited precision and performance costs due to run-time conversions might result.

  If possible, use the following data types:
  – Character instead of varying character for short columns
  – Integer instead of float or decimal
  – Datetime instead of character
  – Numeric instead of character
- To decrease the possibility that a sort operation will occur, omit clauses or operations such as DISTINCT or ORDER BY if such operations are not required.
- To check for existence of rows in a table, select a single row. Either open a cursor and fetch one row or perform a a single-row (SELECT INTO) selection. Remember to check for the SQLCODE -811 error if more than one row is found.

  Unless you know that the table is very small, do not use the following statement to check for a non-zero value:

  ```
  SELECT COUNT(*) FROM  TABLENAME
  ```

  For large tables, counting all the rows impacts performance.
- If update activity is low and tables are large, define indexes on columns that are frequently used as predicates.
- Consider using an IN list if the same column appears in multiple predicate clauses. For large IN lists used with host variables, looping a subset of the host variables might improve performance.

The following suggestions apply specifically to SELECT statements that access several tables.

- Use join predicates to join tables. A join predicate is a comparison between two columns from different tables in a join.
- Define indexes on the columns in the join predicate to allow the join to be processed more efficiently. Indexes also benefit UPDATE and DELETE statements that contain SELECT statements that access several tables.
- If possible, avoid using expressions or OR clauses with join predicates because the database manager cannot use some join techniques. As a result, the most efficient join method may not be chosen.
- In a partitioned database environment, if possible ensure that both of the tables joined are partitioned on the join column.

Data sampling in SQL and XQuery queries

Databases are growing so large and queries on those databases so complex that it is often impractical and sometimes unnecessary to access all of the data relevant to a query. In some cases, a user is interested in finding overall trends or patterns, in which case approximate answers within some margin of error will suffice. One way to speed up such queries is to perform the query on a random sample of the database. DB2 allows you to do efficient sampling of data in SQL and XQuery queries, potentially improving performance of large queries by orders of magnitude while maintaining a high degree of accuracy.

The most common application of sampling is for aggregate queries such as AVG, SUM, and COUNT, where reasonably accurate answers of the aggregates can be obtained from a sample of the data. Sampling can also be used to obtain a random subset of the actual rows in a table for auditing purposes or to speed up data mining and analysis tasks.

DB2 provides two methods of sampling: row-level sampling and block-level sampling.

Row-level Bernoulli sampling:

Row-level Bernoulli sampling gets a sample of P percent of the table rows by means of a SARGable predicate that includes each row in the sample with a probability of P/100 and excludes it with a probability of 1–P/100.

Row-level Bernoulli sampling always gets a valid, random sample regardless of data clustering. However, the performance of this type of sampling is very poor if no index is available because every row must be retrieved and the sampling predicate applied to it. If there is no index then there are no I/O savings over executing a query without sampling. If an index is available, then performance using this type of sampling is improved because the sampling predicate is applied on the RIDS inside the index leaf pages. In the usual case, this requires one I/O per selected RID, and one I/O per index leaf page.

System page-level sampling:

System page-level sampling is similar to row-level sampling, except that pages are sampled and not rows. A page is included in the sample with a probability of P/100. If a page is included, all of the rows in that page are included.

Performance of system page-level sampling is excellent because only one I/O is required for each page that is included in the sample. Compared with no sampling, page-level sampling improves performance by orders of magnitude. However, the accuracy of aggregate estimates tends to be worse under page-level sampling than row-level sampling. This disparity in accuracy is most pronounced when there are many rows per block or when the columns referenced in the query exhibit a high degree of clustering within the pages.

The best sampling method for a particular task will be determined by a user's time constraints and the desired degree of accuracy.

Specifying the sampling method:

To execute a query on a random sample of data from a table, you can use the TABLESAMPLE clause of the table-reference clause in a SQL statement. To specify the method of sampling, use the keywords BERNOULLI or SYSTEM.

The BERNOULLI keyword specifies that row-level Bernoulli sampling is performed.

The SYSTEM keyword specifes that system page-level sampling is performed unless the optimizer determines that it is more efficient to perform row-level Bernoulli sampling instead.

Related reference:

- "Subselect" in *SQL Reference, Volume 1*

Tuning sort performance

Because queries often require sorted or grouped results, sorting is often required, and the proper configuration of the sort heap areas is crucial to good query performance. Sorting is required when:

- No index exists to satisfy a requested ordering (for example a SELECT statement that uses the ORDER BY clause).
- An index exists but sorting would be more efficient than using the index
- An index is created.
- An index is dropped, which causes index page numbers to be sorted.

Elements that affect sorting

The following factors affect sort performance:

- The settings for the following database configuration parameters:
  - Sort heap size, (*sortheap*), which specifies the amount of memory to be used for each sort
  - Sort heap threshold (*sheapthres*) and the sort heap threshold for shared sorts (*sheapthres_shr*), which control the total amount of memory for sorting available across the entire instance for all sorts
- The number of statements in a workload which require a large amount of sorting.
- The existence or absence of indexes that could help avoid unnecessary sorting
- The use of application logic that does not minimize sorting
- Parallel sorting, which improves the performance of sorts but can only occur if the statement uses intra-partition parallelism.
- Whether the sort is *overflowed* or *non-overflowed*. If the sorted data cannot fit entirely into the sort heap, which is a block of memory that is allocated each timea sort is performed, it overflows into a temporary table owned by the database. Sorts that do not overflow always perform better than those that do.
- Whether the results of the sort are *piped* or *non-piped*. If sorted information can return directly without requiring a temporary table to store a final, sorted list of data, it is a piped sort. If the sorted information requires a temporary table to be returned, it is a non-piped sort. A piped sort always performs better than a non-piped sort.

  Also note that in a piped sort, the sort heap is not freed until the application closes the cursor associated with that sort. A piped sort can continue to use up memory until the cursor is closed.

In general, overall sort memory available across the instance (*sheapthres*) should be as large as possible without causing excessive paging. Although a sort can be performed entirely in sort memory, this might cause excessive page swapping. In this case, you lose the advantage of a large sort heap. For this reason, you should use an operating system monitor to track changes in system paging whenever you adjust the sorting configuration parameters.

Techniques for managing sorting performance

Identify particular applications and statements where sorting is a significant performance problem:

1. Set up event monitors at the application and statement level to help you identify applications with the longest total sort time.

2. Within each of these applications, find the statements with the longest *total sort time*.

   You can also search through the explain tables to identify the queries that have sort operations.
3. Use these statements as input to the Design Advisor, which will identify and optionally create indexes to reduce the need for sorting.

Use the database system monitor and benchmarking techniques to help set the *sortheap* and *sheapthres* configuration parameters. For each database manager and its databases:

1. Set up and run a representative workload.
2. For each applicable database, collect average values for the following performance variables over the benchmark workload period:
   - Total sort heap in use
   - Active sorts
3. Set *sortheap* to the average *total sort heap in use* for each database.

   Note: With the improvement in the DB2 partial-key binary sorting technique to include non-integer data type keys, some additional memory is required when sorting long keys. If long keys are used for sorts, you may need to increase the *sortheap* configuration parameter.
4. Set the *sheapthres*. To estimate an appropriate size:
   a. Determine which database in the instance has the largest *sortheap* value.
   b. Determine the average size of the sort heap for this database.

      If this is too difficult to determine, use 80% of the maximum sort heap
   c. Set *sheapthres* to the average number of active sorts times the average size of the sort heap computed above.

      This is a recommended initial setting. You can then use benchmark techniques to refine this value.

Related reference:

- "sortheap - Sort heap size " on page 400
- "sheapthres - Sort heap threshold " on page 399
- "sheapthres_shr - Sort heap threshold for shared sorts " on page 390

Buffer pools and prefetching

Buffer pool management

A buffer pool is memory used to cache table and index data pages as they are being read from disk, or being modified. The buffer pool improves database system performance by allowing data to be accessed from memory instead of from disk. Because memory access is much faster than disk access, the less often the database manager needs to read from or write to a disk, the better the performance. Because most data manipulation takes place in buffer pools, configuring buffer pools is the single most important tuning area. Only large objects and long field data are not manipulated in a buffer pool.

When an application accesses a row of a table for the first time, the database manager places the page containing that row in the buffer pool. The next time any

application requests data, the database manager looks for it in the buffer pool. If the requested data is in the buffer pool, it can be retrieved without disk access, resulting in faster performance.

Memory is allocated for the buffer pool when a database is activated or when the first application connects to the database. Buffer pools can also be created, dropped, and resized while the database is manager is running. If you use the IMMEDIATE keyword when you use the ALTER BUFFERPOOL statement to increase the size of the buffer pool, memory is allocated as soon as you enter the command if the memory is available. If the memory is unavailable, the change occurs when all applications are disconnected and the database is reactivated. If you decrease the size of the buffer pool, memory is deallocated at commital time. When all applications are disconnected, the buffer pool memory is de-allocated.

Note: To reduce the necessity of increasing the size of the *dbheap* database configuration parameter when buffer pool sizes increase, nearly all buffer pool memory, which includes page descriptors, buffer pool descriptors, and the hash tables, comes out of the database shared memory set and is sized automatically.

To ensure that an appropriate buffer pool is available in all circumstances, DB2 creates small system buffer pools, one with each page size: 4K, 8K, 16K, and 32K. The size of each buffer pool is 16 pages. These buffer pools are hidden from the user. They are not present in the system catalogs or in the buffer pool system files. You cannot use or alter them directly, but DB2 uses these buffer pools in the following circumstances:

- When a buffer pool of the required page size is inactive because insufficient memory was available to create it after a CREATE BUFFERPOOL statement was executed with the IMMEDIATE keyword.

  A message is written to the administration notification log. If necessary, table spaces are remapped to a system buffer pool. Performance might be drastically reduced.
- When the ordinary buffer pools cannot be brought up during a database connect

  This problem is likely to have a serious cause, such as out-of-memory condition. Although DB2 will be fully functional because of the system buffer pools, performance will degrade drastically. You should address this problem immediately. You receive a warning when this occurs and a message is written to the administration notification log.

Pages remain in the buffer pool until the database is shut down, or until the space occupied by a page is required for another page. The following criteria determine which page is removed to bring in another page:

- How recently the page was referenced
- The probability that the page will be referenced again by the last agent that looked at it
- The type of data on the page
- Whether the page was changed in memory but not written out to disk (Changed pages are always written to disk before being overwritten.)

In order for pages to be accessed from memory again, changed pages are not removed from the buffer pool after they are written out to disk unless the space is needed.

When you create a buffer pool, the default page size is the size specified when the database was created unless you explicitly specify a page size other than the default page size. Because pages can be read into a buffer pool only if the table space page size is the same as the buffer pool page size, the page size of your table spaces should determine the page size that you specify for buffer pools. You cannot alter the page size of the buffer pool after you create it. You must create a new buffer pool with a different page size.

The memory tracker, invoked by db2mtrk, allows you to view the amount of database memory allocated to the buffer pools. The following sample db2mtrk output shows one user-created buffer pool, identified by the number ″1″ in parentheses, and four system buffer pools, identified by the page size in parentheses:

```
> db2mtrk -d
Tracking Memory on: 2005/10/24 at 12:47:26

Memory for database: XMLDB

   utilh     pckcacheh catcacheh bph (1)   bph (S32K)  bph (S16K)  bph (S8K)
   64.0K     576.0K    64.0K     4.2M      576.0K      320.0K      192.0K

   bph (S4K)  shsorth   lockh     dbh       other
   128.0K     0         640.0K    4.2M      192.0K
```

Note: On 32-bit platforms that run Windows, you can create large buffer pools if you have enabled Address Windowing Extensions (AWE) or Advanced Server and Data Center Server on Windows.

Related concepts:

- “Buffer pool management of data pages” on page 116
- “Illustration of buffer pool data-page management” on page 118
- “Management of multiple database buffer pools” on page 114
- “Memory allocation in DB2” on page 247

Related reference:

- “db2mtrk - Memory tracker command” in *Command Reference*

Management of multiple database buffer pools

Although each database requires at least one buffer pool, you can create several buffer pools, each of a different size or with a different page size, for a single database that has table spaces of more than one page size. Each buffer pool has a minimum size, depending on the platform. You can use the ALTER BUFFERPOOL command to resize a buffer pool.

A new database has a default buffer pool called IBMDEFAULTBP with an overall size determined by the platform and a default page size based on the page size specified when the database was created. The default page size is stored as an informational database configuration parameter called *pagesize*. When you create a table space with the default page size and you do not assign it to a specific buffer pool, the table space is assigned to the default buffer pool. You can resize the default buffer pool and change its attributes, but you cannot drop it.

Page sizes for buffer pools

After you create or migrate a database, you can create additional buffer pools. You can create the database with 8 KB page size as the default and the default buffer pool will be created with the default page size (in this case, an 8 KB page size). Alternatively, you can create a buffer pool with an 8 KB page size as well as one or more table spaces with the same page size. This method does not require that you change the 4 KB default page size when you create the database. You cannot assign a table space to a buffer pool that uses a different page size.

Note: If you create a table space with a page size greater than 4 KB, such as 8 KB, 16 KB, or 32 KB, you need to assign it to a buffer pool that uses the same page size. If this buffer pool is currently not active, DB2 attempts to assign the table space temporarily to another active buffer pool that uses the same page size if one exists or to one of the default system buffer pools that DB2 creates when the first client connects to the database. When the database is activated again, and the originally specified buffer pool is active, then DB2 assigns the table space to that buffer pool.

When you create a buffer pool, you specify the size of the buffer pool. To increase or decrease the buffer-pool size later, use the ALTER BUFFERPOOL statement.

In a partitioned database environment, each buffer pool for a database has the same default definition on all database partitions unless it was otherwise specified in the CREATE BUFFERPOOL statement, or the buffer-pool size was changed by the ALTER BUFFERPOOL statement for a particular database partition.

Advantages of large buffer pools

Large buffer pools provide the following advantages:

- They enable frequently requested data pages to be kept in the buffer pool, which allows quicker access. Fewer I/O operations can reduce I/O contention, thereby providing better response time and reducing the processor resource needed for I/O operations.
- They provide the opportunity to achieve higher transaction rates with the same response time.
- They prevent I/O contention for frequently used disk storage devices such as catalog tables and frequently referenced user tables and indexes. Sorts required by queries also benefit from reduced I/O contention on the disk storage devices that contain the temporary table spaces.

Advantages of many buffer pools

If any of the following conditions apply to your system, you should use only a single buffer pool:

- The total buffer space is less than 10 000 4 KB pages.
- People with the application knowledge to do specialized tuning are not available.
- You are working on a test system.

In all other circumstances, consider using more than one buffer pool for the following reasons:

- Temporary table spaces can be assigned to a separate buffer pool to provide better performance for queries that require temporary storage, especially sort-intensive queries.

- If data must be accessed repeatedly and quickly by many short update-transaction applications, consider assigning the table space that contains the data to a separate buffer pool. If this buffer pool is sized appropriately, its pages have a better chance of being found, contributing to a lower response time and a lower transaction cost.
- You can isolate data into separate buffer pools to favor certain applications, data, and indexes. For example, you might want to put tables and indexes that are updated frequently into a buffer pool that is separate from those tables and indexes that are frequently queried but infrequently updated. This change will reduce the impact that frequent updates on the first set of tables have on frequent queries on the second set of tables.
- You can use smaller buffer pools for the data accessed by applications that are seldom used, especially for an application that requires very random access into a very large table. In such a case, data need not be kept in the buffer pool for longer than a single query. It is better to keep a small buffer pool for this data, and free the extra memory for other uses, such as for other buffer pools.
- After separating different activities and data into separate buffer pools, good and relatively inexpensive performance diagnosis data can be produced from statistics and accounting traces.

Buffer pool memory allocation at startup

When you use the CREATE BUFFERPOOL command to create a buffer pool or use the ALTER BUFFERPOOL statement to alter buffer pools, the total memory that is required by all buffer pools must be available to the database manager so all of the buffer pools can be allocated when the database is started. If you create or modify buffer pools while the database manager is online, additional memory should be available in database global memory. If you specify the IMMEDIATE keyword when you create a new buffer pool or increase the size of an existing buffer pool and the required memory is unavailable, the database manager makes the change the next time the database is activated. On 32-bit platforms, the memory must be available and can be reserved in the global database memory, as described in detailed information for the *database_memory* database configuration parameter.

If this memory is not available when a database starts, the database manager will only start with system buffer pools (one for each page size) with a minimal size of 16 pages, and an SQL1478W (SQLSTATE01626) warning is returned. The database continues in this operational state until its configuration is changed and the database can be fully restarted. Performance may be suboptimal. The database manager starts with minimal-sized values only to allow you to connect to the database so that you can reconfigure the buffer pool sizes or perform other critical tasks. As soon as you perform these tasks, restart the database. Do not operate the database for an extended time in such a state.

To avoid starting the database with system buffer pools only, you can use the DB2_OVERRIDE_BPF registry variable to override the current buffer pool sizes so that the total memory required is available.

Related concepts:

- "Buffer pool management" on page 112

Buffer pool management of data pages

Pages in the buffer pool can be either *in-use* or not, and they can be *dirty* or *clean*:

- In-use pages are currently being read or updated. While a page is in use by an agent, it can be read, but not updated, by other agents or prefetchers in the database.
- "Dirty" pages contain data that has been changed but has not yet been written to disk.
- After a changed page is written to disk, it is clean but remains in the buffer pool until its space is needed for new pages.

Page-cleaner agents

In a well-tuned system, it is usually the page-cleaner agents that write changed or "dirty" pages to disk. Page-cleaner agents perform I/O as background processes and allow applications to run faster because their agents can perform actual transaction work. Page-cleaner agents are sometimes referred to as *asynchronous page cleaners* or *asynchronous buffer writers* because they are not coordinated with the work of other agents and work only when required.

To improve performance in update-intensive workloads, you might want to configure more page-cleaner agents. Performance can improve if more page-cleaner agents are available to write dirty pages to disk. This is particularly true if snapshots reveal that there are a significant number of synchronous data-page or index-page writes in relation to the number of asynchronous data-page or index-page writes.

Page cleaning and fast recovery

If more pages have been written to disk, recovery of the database is faster after a system crash because the database manager can rebuild more of the buffer pool from disk instead of having to replay transactions from the database log files.

The size of the log that must be read during recovery is the difference between the location of the following records in the log:

- The most recently written log record
- The log record that describes the oldest change to data in the buffer pool.

The default behavior of the page cleaners is that page cleaning is performed if the size of the log that would need to be replayed during recovery exceeds the following maximum:

```
logfilsiz * softmax
```

where:

- *logfilsiz* represents the size of the log files
- *softmax* represents the percentage of log files to be recovered following a database crash. For example, if the value of *softmax* is 250, then 2.5 log files will contain the changes that need to be recovered if a crash occurs.

To minimize log read time during recovery, use the database system monitor to track the number of times that page cleaning is performed. The system monitor *pool_lsn_gap_clns (buffer pool log space cleaners triggered)* monitor element provides this information if you have not enabled proactive page cleaning for your database. If you have enabled this alternate page cleaning, this condition should not occur and the *pool_lsn_gap_clns* monitor element is always 0.

The *log_held_by_dirty_pages* monitor element can be used to determine if the page cleaners are not cleaning enough pages to meet the recovery criteria set by the user. If *log_held_by_dirty_pages* is consistently and significantly greater than *logfilsiz* * *softmax*, then either more page cleaners are required, or *softmax* needs to be adjusted.

Related concepts:

- "Illustration of buffer pool data-page management" on page 118

Illustration of buffer pool data-page management

The following figure illustrates how the work of managing the buffer pool can be shared between page-cleaner agents and database agents, compared to the database agents performing all of the I/O.

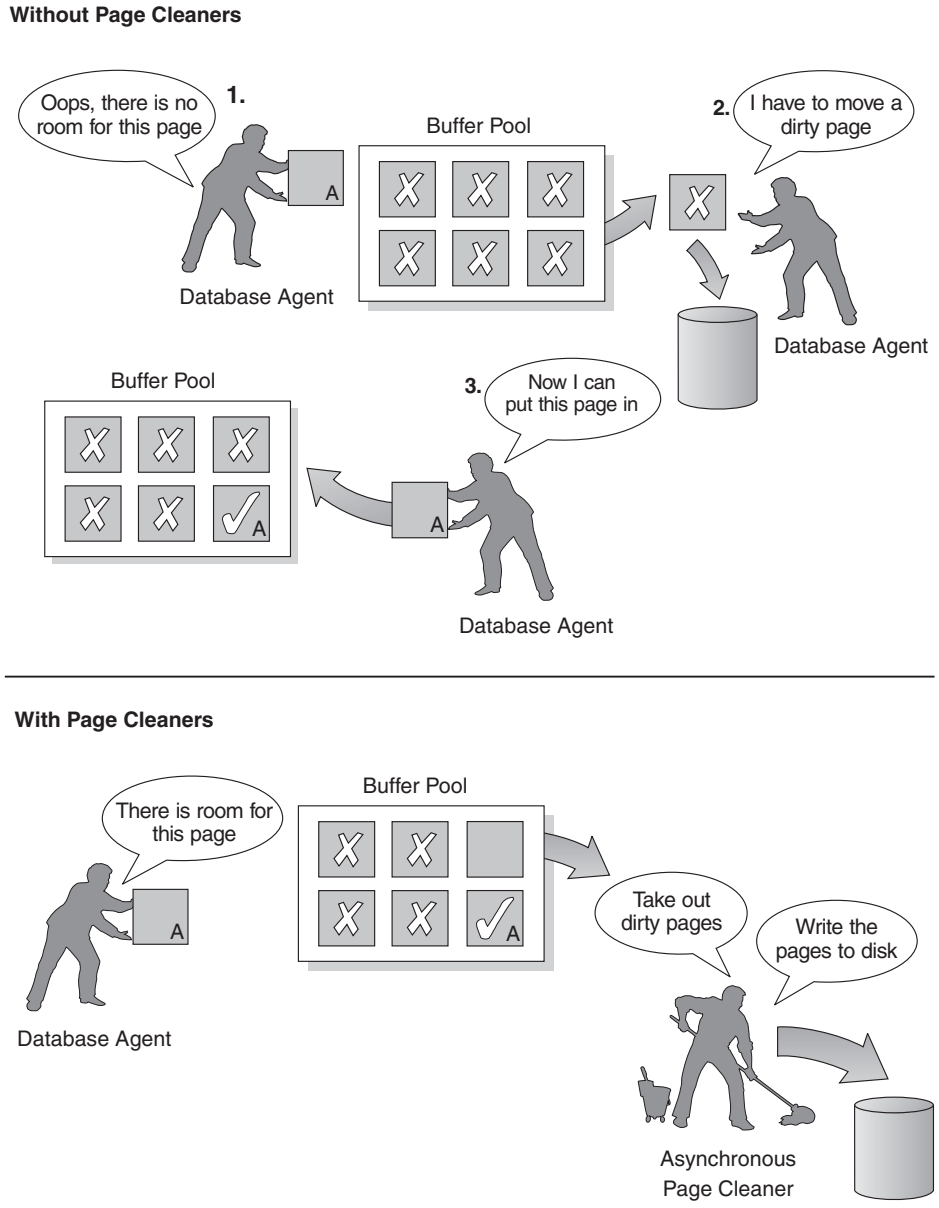


Figure 12. Asynchronous page cleaner. "Dirty" pages are written out to disk.

Related concepts:

- "Buffer pool management" on page 112
- "Buffer pool management of data pages" on page 116

Update processing

When an agent updates a page, the database manager uses the following protocol to minimize the I/O required by the transaction and ensure recoverability.

1. The page to be updated is pinned and latched with an exclusive lock. A log record is written to the log buffer describing how to redo and undo the change. As part of this action, a log sequence number (LSN) is obtained and is stored in the page header of the page being updated.
2. The change is made to the page.
3. The page is unlatched and unfixed.

   The page is considered to be "dirty" because changes to the page have not been written out to disk.
4. The log buffer is updated.

   Both the data in the log buffer and the "dirty" data page are forced to disk.

For better performance, these I/Os are delayed until a convenient point, such as during a lull in the system load, or until necessary to ensure recoverability, or to limit recovery time. Specifically, a "dirty" page is forced to disk at the following times:

- When another agent chooses it as a victim.
- When a page cleaner acts on the page as the result of:
  - Another agent choosing it as a victim.
  - The *chngpgs_thresh* database configuration parameter percentage value is exceeded. When this value is exceeded, asynchronous page cleaners wake up and write changed pages to disk.

    If proactive page cleaning is enabled, this value is irrelevant and does not trigger page cleaning.
  - The *softmax* database configuration parameter percentage value is exceeded. Once exceeded, asynchronous page cleaners wake up and write changed pages to disk.

    If proactive page cleaning is enabled for the database, and the number of page cleaners has been properly configured for the database, this value should never be exceeded.
  - The number of clean pages on the hate list drops too low. Page cleaners only react to this condition under the proactive page cleaning method.
  - When a dirty page currently contributes to, or is projected to contribute to an LSNGAP condition. Page cleaners only react to this condition under the proactive page cleaning method.
- When the page was updated as part of a table which has the NOT LOGGED INITIALLY clause invoked and a COMMIT statement is issued. When the COMMIT statement is executed, all changed pages are flushed to disk to ensure recoverability.

Related concepts:

- "Reducing logging overhead to improve query performance" on page 129
- "Client-server processing model" on page 142

Related reference:

- "softmax - Recovery range and soft checkpoint interval " on page 451
- "chngpgs_thresh - Changed pages threshold " on page 416

Proactive page cleaning

Starting in Version 8.1.4, there is an alternate method of configuring page cleaning in your system. This alternate method differs from the default behavior in that page cleaners behave more proactively in choosing which dirty pages get written out at any given point in time. This new method of page cleaning differs from the default page cleaning method in two major ways:

1. Page cleaners do not respect the chngpgs_thresh configuration parameter.

   In this alternative method of page cleaning, page cleaners will no longer react in response to value of the chngpgs_thresh configuration parameter. Instead of attempting to keep some percentage of the buffer pool clean, the alternate method of page cleaning provides a mechanism whereby the agents are informed of the location of good victim pages that have just been written out, so that agents do not have to search the buffer pool to look for a victim. When the number of good victim pages drops below an acceptable value, the page cleaners are triggered, and proceed to search the entire buffer pool, writing out potential victim pages, and informing the agents of the location of these pages.
2. Page cleaners no longer respond to LSN gap triggers issued by the logger.

   When the amount of log space encompassing the log record which has updated the oldest page in the buffer pool and the current log position exceeds that allowed by the softmax parameter, it is said that the database is in an 'LSN gap' situation. Under the default method of page cleaning, when the logger detects than an LSN gap has occurred, it will trigger the page cleaners to write out all the pages which are contributing to the LSN gap situation. That is, it will write out those pages which are older than what is allowed by the softmax parameter. Page cleaners will be idle for some period of time while no LSN gap is occurring. Then, once an LSN gap occurs, the page cleaners are activated to write a large number of pages before going back to sleep. This can result in the saturation of the I/O subsystem, which then affects other agents which are reading or writing pages. Furthermore, by the time an LSN gap is triggered, it is possible that the page cleaners will not be able to clean fast enough and DB2 might run out of log space.

   The alternate method of page cleaning modulates this behavior by spreading out the same number of writes over a greater period of time. The cleaners do this by proactively cleaning not only pages the pages that are currently in an LSN gap situation, but also the pages that will come into an LSN gap situation soon, based on the current level of activity.

To use the new method of page cleaning, set the DB2_USE_ALTERNATE_PAGE_CLEANING registry variable to "ON".

Related concepts:

- "Buffer pool management of data pages" on page 116

Prefetching data into the buffer pool

Prefetching pages means that one or more pages are retrieved from disk in the expectation that they will be required by an application. Prefetching index and data pages into the buffer pool can help improve performance by reducing the I/O wait time. In addition, parallel I/O enhances prefetching efficiency.

There are two categories of prefetching:

- **Sequential prefetch:** A mechanism that reads consecutive pages into the buffer pool before the pages are required by the application.
- **List prefetch:** Sometimes called *list sequential prefetch*. Prefetches a set of non-consecutive data pages efficiently.

These two methods of reading data pages are in addition to a normal read. A normal read is used when only one or a few consecutive pages are retrieved. During a normal read, one page of data is transferred.

Prefetching and Intra-Partition Parallelism

Prefetching is important to the performance of intra-partition parallelism, which uses multiple subagents when scanning an index or a table. Such parallel scans introduce larger data-consumption rates, which require higher prefetch rates.

The cost of inadequate prefetching is higher for parallel scans than serial scans. If prefetching does not occur for a serial scan, the query runs more slowly because the agent always needs to wait for I/O. If prefetching does not occur for a parallel scan, all subagents might need to wait because one subagent is waiting for I/O.

Because of its importance, prefetching is performed more aggressively with intra-partition parallelism. The sequential detection mechanism tolerates larger gaps between adjacent pages so that the pages can be considered sequential. The width of these gaps increases with the number of subagents involved in the scan.

Related concepts:

- "Buffer pool management" on page 112
- "I/O server configuration for prefetching and parallelism" on page 124
- "Illustration of prefetching with parallel I/O" on page 125
- "List prefetching" on page 124
- "Sequential prefetching" on page 121

Sequential prefetching

Reading several consecutive pages into the buffer pool using a single I/O operation can greatly reduce your application overhead. In addition, multiple parallel I/O operations to read several ranges of pages into the buffer pool can help reduce I/O wait time.

Prefetching starts when the database manager determines that sequential I/O is appropriate and that prefetching might improve performance. In cases such as table scans and table sorts, the database manager can easily determine that sequential prefetch will improve I/O performance. In these cases, the database manager automatically starts sequential prefetch. The following example, which probably requires a table scan, would be a good candidate for sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
```

Implications of the PREFETCHSIZE for table spaces

To define the number of prefetched pages for each table space, use the PREFETCHSIZE clause in either the CREATE TABLESPACE or ALTER TABLESPACE statements. The value that you specify is maintained in the PREFETCHSIZE column of the SYSCAT.TABLESPACES system catalog table.

It is a good practice to explicitly set the PREFETCHSIZE value as a multiple of the number of table space containers, the number of physical disks under each container (if a RAID device is used) and the EXTENTSIZE value for your table space, which is the number of pages that the database manager writes to a container before it uses a different container. For example, if the extent size is 16 pages and the table space has two containers, you might set the prefetch quantity to 32 pages. If there are 5 physical disks per container, then you might set the prefetch quantity to 160 pages.

The database manager monitors buffer-pool usage to ensure that prefetching does not remove pages from the buffer pool if another unit of work needs them. To avoid problems, the database manager can limit the number of prefetched pages to less than you specify for the table space.

The prefetch size can have significant performance implications, particularly for large table scans. Use the database system monitor and other system monitor tools to help you tune PREFETCHSIZE for your table spaces. You might gather information about whether:

- There are I/O waits for your query, using monitoring tools available for your operating system.
- Prefetch is occurring, by looking at the *pool_async_data_reads (buffer pool asynchronous data reads)* data element provided by the database system monitor.

If there are I/O waits and the query is prefetching data, you might increase the value of PREFETCHSIZE. If the prefetcher is not the cause of the I/O wait, increasing the PREFETCHSIZE value will not improve the performance of your query.

In all types of prefetch, multiple I/O operations might be performed in parallel when the prefetch size is a multiple of the extent size for the table space and the extents of the table space are in separate containers. For better performance, configure the containers to use separate physical devices.

Sequential detection

In some cases it is not immediately obvious that sequential prefetch will improve performance. In these cases, the database manager can monitor I/O and activate prefetching if sequential page reading is occurring. In this case, prefetching is activated and deactivated by the database manager as appropriate. This type of sequential prefetch is known as *sequential detection* and applies to both index and data pages. Use the *seqdetect* configuration parameter to control whether the database manager performs sequential detection.

For example, if sequential detection is turned on, the following SQL statement might benefit from sequential prefetch:

```
SELECT NAME FROM EMPLOYEE
WHERE EMPNO BETWEEN 100 AND 3000
```

In this example, the optimizer might have started to scan the table using an index on the EMPNO column. If the table is highly clustered with respect to this index, then the data-page reads will be almost sequential and prefetching might improve performance, so data-page prefetch will occur.

Index-page prefetch might also occur in this example. If many index pages must be examined and the database manager detects that sequential page reading of the index pages is occurring, then index-page prefetching occurs.

Related concepts:

- "Block-based buffer pools for improved sequential prefetching" on page 123
- "Buffer pool management" on page 112
- "List prefetching" on page 124
- "Prefetching data into the buffer pool" on page 120

Block-based buffer pools for improved sequential prefetching

Prefetching pages from disk is expensive because of I/O overhead. Throughput can be significantly improved if processing is overlapped with I/O. Most platforms provide high-performance primitives that read contiguous pages from disk into non-contiguous portions of memory. These primitives are usually called *scattered read* or *vectored I/O*. On some platforms, performance of these primitives cannot compete with doing I/O in large block sizes.

By default, the buffer pools are page-based, which means that contiguous pages on disk are prefetched into non-contiguous pages in memory. Sequential prefetching can be enhanced if contiguous pages can be read from disk into contiguous pages within a buffer pool.

You can create block-based buffer pools for this purpose. A block-based buffer pool consist of both a page area and a block area. The page area is required for non-sequential prefetching workloads. The block area consist of blocks where each block contains a specified number of contiguous pages, which is referred to as the *block size*.

The optimal usage of a block-based buffer pool depends on the specified block size. The block size is the granularity at which I/O servers doing sequential prefetching consider doing block-based I/O. The extent is the granularity at which table spaces are striped across containers. Because multiple table spaces with different extent sizes can be bound to a buffer pool defined with the same block size, consider how the extent size and the block size interact for efficient use of buffer-pool memory. Buffer-pool memory can be wasted in the following circumstances:

- If the extent size, which determines the prefetch request size, is smaller than BLOCK_SIZE specified for the buffer pool.
- If some pages requested in the prefetch request are already present in the page area of the buffer pool.

The I/O server allows some wasted pages in each buffer-pool block, but if too much of a block would be wasted, the I/O server does non-block-based prefetching into the page area of the buffer pool. This is not optimal performance.

For optimal performance, bind table spaces of the same extent size to a buffer pool with a block size that equals the table-space extent size. Good performance can be achieved if the extent size is larger than the block size, but not when the extent size is smaller than the block size.

To create block-based buffer pools, use the CREATE and ALTER BUFFERPOOL statements.

Block-based I/O and AWE support cannot be used by a buffer pool simultaneously. AWE support takes precedence over block-based I/O support when both are enabled for a given buffer pool. In this situation, the block-based I/O support is disabled for the buffer pool. It is re-enabled when the AWE support is disabled.

Note: Block-based buffer pools are intended for sequential prefetching. If your applications do not use sequential prefetching, then the block area of the buffer pool is wasted.

Related concepts:

- "Buffer pool management" on page 112
- "Prefetching data into the buffer pool" on page 120
- "Sequential prefetching" on page 121

List prefetching

List prefetch, or *list sequential prefetch*, is a way to access data pages efficiently even when the data pages needed are not contiguous. List prefetch can be used in conjunction with either single or multiple index access.

If the optimizer uses an index to access rows, it can defer reading the data pages until all the row identifiers (RIDs) have been obtained from the index. For example, the optimizer could perform an index scan to determine the rows and data pages to retrieve, given the previously defined index IX1:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

and the following search criteria:

```
WHERE NAME BETWEEN 'A' and 'I'
```

If the data is not clustered according to this index, list prefetch includes a step that sorts the list of RIDs obtained from the index scan.

Related concepts:

- "Buffer pool management" on page 112
- "Prefetching data into the buffer pool" on page 120
- "Sequential prefetching" on page 121

I/O server configuration for prefetching and parallelism

To enable prefetching, the database manager starts separate threads of control, known as *I/O servers*, to read data pages. As a result, the query processing is divided into two parallel activities: data processing (CPU) and data page I/O. The I/O servers wait for prefetch requests from the CPU processing activity. These prefetch requests contain a description of the I/O needed to satisfy the query. The possible prefetch methods determine when and how the database manager generates the prefetch requests.

Configuring enough I/O servers with the *num_ioservers* configuration parameter can greatly enhance the performance of queries for which prefetching of data can

be used. To maximize the opportunity for parallel I/O, set *num_ioservers* to at least the number of physical disks in the database.

It is better to overestimate the number of I/O servers than to underestimate. If you specify extra I/O servers, these servers are not used, and their memory pages are paged out. As a result, performance does not suffer. Each I/O server process is numbered. The database manager always uses the lowest numbered process, so some of the upper numbered processes might never be used.

To estimate the number of I/O servers that you might need, consider the following:

- The number of database agents that could be writing prefetch requests to the I/O server queue concurrently.
- The highest degree to which the I/O servers can work in parallel.

It is worthwhile to consider setting the value of the *num_ioservers* database configuration parameter to AUTOMATIC so that DB2 can pick intelligent values based on the system configuration.

Configuration for asynchronous I/O

On some platforms, DB2 uses asynchronous I/O (AIO) to improve performance of activities such as page cleaning and prefetching. AIO is most effective if data in containers is distributed across multiple disks. Performance also benefits from tuning the underlying operating system AIO infrastructure.

For example, on AIX®, you might tune AIO on the operating system. When AIO works on either SMS or DMS file containers, operating system processes called AIO servers manage the I/O. A small number of such servers might restrict the benefit of AIO by limiting the number of AIO requests. To configure the number of AIO servers on AIX, use the *smit* AIO *minservers* and *maxservers* parameters.

Related concepts:

- "Illustration of prefetching with parallel I/O" on page 125
- "Parallel I/O management" on page 127

Related reference:

- "num_ioservers - Number of I/O servers " on page 420

Illustration of prefetching with parallel I/O

The following figure illustrates how I/O servers are used to prefetch data into a buffer pool.

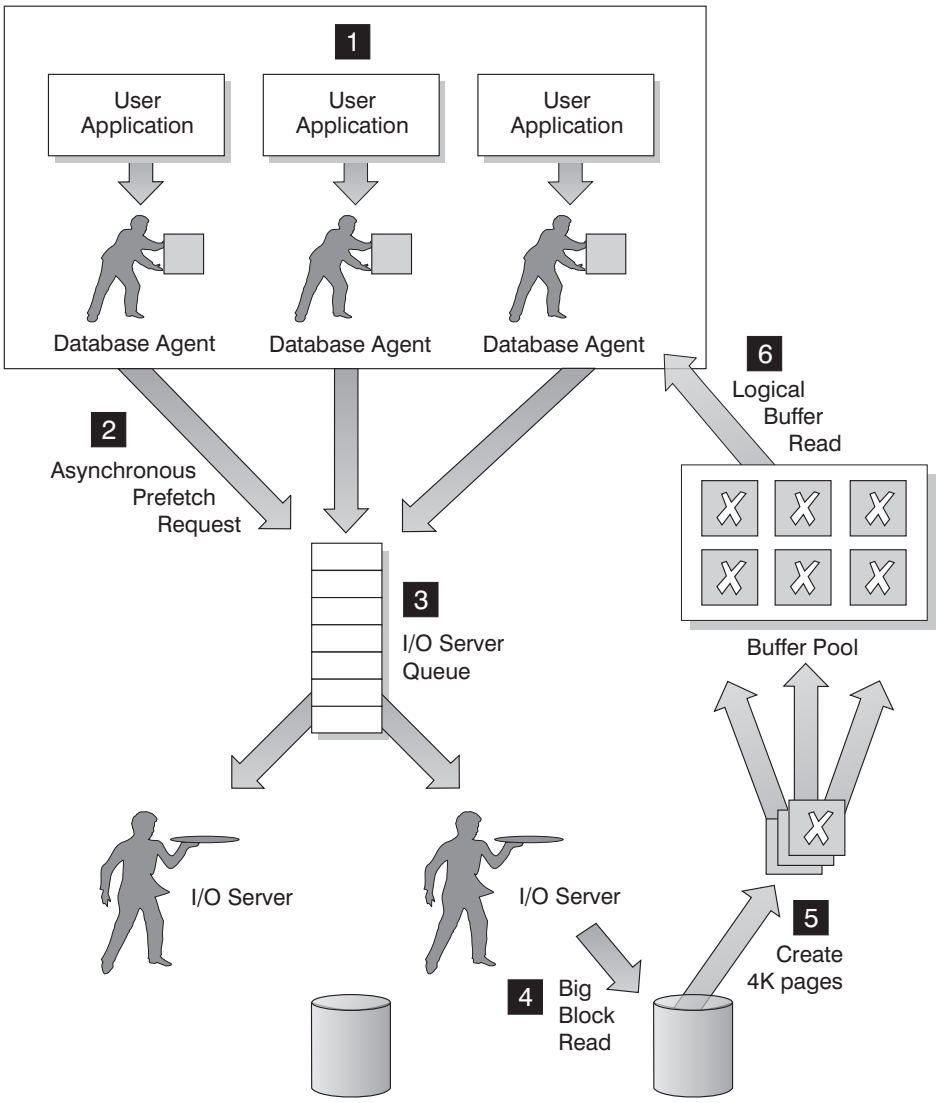


Figure 13. Prefetching data using I/O servers

1 The user application passes the request to the database agent that has been assigned to the user application by the database manager.

2, **3**
The database agent determines that prefetching should be used to obtain the data required to satisfy the request and writes a prefetch request to the I/O server queue.

4, **5**
The first available I/O server reads the prefetch request from the queue and then reads the data from the table space into the buffer pool. The number of I/O servers that can fetch data from a table space at the same time depends on the number of prefetch requests in the queue and the number of I/O servers configured by the *num_ioservers* database configuration parameter.

6 The database agent performs the necessary operations on the data pages in the buffer pool and returns the result to the user application.

Related concepts:

- "Agents in a partitioned database" on page 140
- "Parallel I/O management" on page 127
- "I/O server configuration for prefetching and parallelism" on page 124
- "List prefetching" on page 124
- "Prefetching data into the buffer pool" on page 120
- "Sequential prefetching" on page 121

Parallel I/O management

If multiple containers exist for a table space, the database manager can initiate *parallel I/O*, in which database manager uses multiple I/O servers to process the I/O requirements of a single query. Each I/O server processes the I/O workload for a separate container, so that several containers can be read in parallel. Performing I/O in parallel can result in significant improvements to I/O throughput.

Although a separate I/O server can handle the workload for each container, the actual number of I/O servers that can perform I/O in parallel is limited to the number of physical devices over which the requested data is spread. For this reason, you need as many I/O servers as physical devices.

Parallel I/O is initiated differently in the following cases:

- **Sequential prefetch**

  For sequential prefetch, parallel I/O is initiated when the prefetch size is a multiple of the extent size for a table space. Each prefetch request is then broken into many small requests along the extent boundaries. These small requests are then assigned to different I/O servers.
- **List prefetch**

  For list prefetch, each list of pages is divided into smaller lists according to the container in which the data pages are stored. These smaller lists are then assigned to different I/O servers.
- **Database or table space backup and restore**

  For backing up or restoring data, the number of parallel I/O requests are equal to the backup buffer size divided by the extent size up to a maximum value equal to the number of containers.
- **Database or table space restore**

  For restoring data, the parallel I/O requests are initiated and split the same way as that used for sequential prefetch. Instead of restoring the data into the buffer pool, the data is moved directly from the restore buffer to disk.
- **Load**

  When you load data, you can specify the level of I/O parallelism with the LOAD command DISK_PARALLELISM option. If you do not specify this option, the database manager uses a default value based on the cumulative number of table space containers for all table spaces associated with the table.

For optimal performance of parallel I/O, ensure that:

- There are enough I/O servers. Specify slightly more I/O servers than the number of containers used for all table spaces within the database.
- The extent size and prefetch size are sensible for the table space. To prevent over-use of the buffer pool, prefetch size should not be too large. An ideal size is a multiple of the extent size, the number of physical disks under each container

(if a RAID device is used) and the number of table space containers. The extent size should be fairly small, with a good value being in the range of 8 to 32 pages.

- The containers reside on separate physical drives.
- All containers are the same size to ensure a consistent degree of parallelism.

  If one or more containers are smaller than the others, they reduce the potential for optimized parallel prefetch. Consider the following examples:

  – After a smaller container is filled, additional data is stored in the remaining containers, causing the containers to become unbalanced. Unbalanced containers reduce the performance of parallel prefetching, because the number of containers from which data can be prefetched might be less than the total number of containers.
  – If a smaller container is added at a later date and the data is rebalanced, the smaller container will contain less data than the other containers. Its small amount of data relative to the other containers will not optimize parallel prefetching.
  – If one container is larger and all of the other containers fill up, it is the only container to store additional data. The database manager cannot use parallel prefetch to access this additional data.
- There is adequate I/O capacity when using intra-partition parallelism. On SMP machines, intra-partition parallelism can reduce the elapsed time for query by running the query on multiple processors. Sufficient I/O capacity is required to keep each processor busy. Usually additional physical drives are required to provide the I/O capacity.

  The prefetch size must be larger for prefetching to occur at higher rates and use I/O capacity effectively.

  The number of physical drives required depends on the speed and capacity of the drives and the I/O bus and on the speed of the processors.

Related concepts:

- "Tuning sort performance" on page 111
- "I/O server configuration for prefetching and parallelism" on page 124
- "Illustration of prefetching with parallel I/O" on page 125

Improving insert performance

When SQL statements use INSERT to place new information in a table, an INSERT search algorithm first searches the Free Space Control Records (FSCRs) to find a page with enough space. However, even when the FSCR indicates a page has enough free space, the space may not be usable because it is reserved by an uncommitted DELETE from another transaction. To ensure that uncommitted free space is usable, you should COMMIT transactions frequently.

The setting of the DB2MAXFSCRSEARCH registry variable determines the number of FSCRs in a table that are searched for an INSERT. The default value for this registry variable is five. If no space is found within the specified number of FSCRs, the inserted record is appended at the end of the table. To optimize INSERT speed, subsequent records are also appended to the end of the table until two extents are filled. After the two extents are filled, the next INSERT resumes searching at the FSCR where the last search ended.

Note: To optimize for INSERT speed at the possible expense of faster table growth, set the DB2MAXFSCRSEARCH registry variable to a small number. To

optimize for space reuse at the possible expense of INSERT speed, set DB2MAXFSCRSEARCH to a larger number.

After all FSCRs in the entire table have been searched in this way, the records to be inserted are appended without additional searching. Searching using the FSCRs is not done again until space is created somewhere in the table, such as following a DELETE.

There are two other INSERT algorithm options, as follows:

- APPEND MODE

  In this mode, new rows are always appended to the end of the table. No searching or maintenance of FSCRs takes place. This option is enabled using the ALTER TABLE APPEND ON statement, and can improve performance for tables that only grow, like journals.
- A clustering index is defined on the table.

  In this case, the database manager attempts to insert records on the same page as other records with similar index key values. If there is no space on that page, the attempt is made to put the record into the surrounding pages. If there is still no success, the FSCR search algorithm, described above, is used, except that a worst-fit approach is used instead of a first-fit approach. This worst-fit approach tends to choose pages with more free space. This method establishes a new clustering area for rows with this key value.

  When you define a clustering index on a table, use ALTER TABLE... PCTFREE before you either load or reorganize the table. The PCTFREE clause specifies the percentage of free space that should remain on the data page of the table after loading and reorganizing. This increases the probability that the cluster index operation will find free space on the appropriate page.

Related concepts:

- "Table and index management for MDC tables" on page 337
- "Table and index management for standard tables" on page 333
- "Update processing" on page 119

Reducing logging overhead to improve query performance

All databases maintain log files that keep records of database changes. There are two logging strategy choices:

- Circular logging, in which the log records fill the log files and then overwrite the initial log records in the initial log file. The overwritten log records are not recoverable.
- Retain log records, in which a log file is archived when it fills with log records. New log files are made available for log records. Retaining log files enables **roll-forward recovery**. Roll-forward recovery reapplies changes to the database based on completed units of work (transactions) that are recorded in the log. You can specify that roll-forward recovery is to the end of the logs, or to a particular point in time before the end of the logs.

Regardless of the logging strategy, all changes to regular data and index pages are written to the log buffer. The data in the log buffer is written to disk by the logger process. In the following circumstances, query processing must wait for log data to be written to disk:

- On COMMIT

- Before the corresponding data pages are written to disk, because DB2 uses write-ahead logging. The benefit of write-ahead logging is that when a transaction completes by executing the COMMIT statement, not all of the changed data and index pages need to be written to disk.
- Before some changes are made to metadata, most of which result from executing DDL statements
- On writing log records into the log buffer, if the log buffer is full

DB2 manages writing log data to disk in this way in order to minimize processing delay. In an environment in which many short concurrent transactions occur, most of the processing delay is caused by COMMIT statements that must wait for log data to be written to disk. As a result, the logger process frequently writes small amounts of log data to disk, with additional delay caused by log I/O overhead. To balance application response time against such logging delay, set the *mincommit* database configuration parameter to a value greater than 1. This setting might cause longer delay for COMMIT from some applications, but more log data might be written in one operation.

Changes to large objects (LOBs) and LONG VARCHARs are tracked through shadow paging. LOB column changes are not logged unless you specify log retain and the LOB column is defined on the CREATE TABLE statement without the NOT LOGGED clause. Changes to allocation pages for LONG or LOB data types are logged like regular data pages.

Related concepts:

- "Client-server processing model" on page 142
- "Update processing" on page 119

Related reference:

- "mincommit - Number of commits to group " on page 449

Chapter 8. Agent Management

Database agents

For each database that an application accesses, various processes or threads start to perform the various application tasks. These tasks include logging, communication, and prefetching.

Database agents are engine dispatchable unit (EDU) processes or threads. Database agents do the work in the database manager that applications request. In UNIX environments, these agents run as processes. In Intel™-based operating systems such Windows, the agents run as threads.

The maximum number of application connections is controlled by the *max_connections* database manager configuration parameter. The work of each application connection is coordinated by a single worker agent.

A *worker agent* carries out application requests but has no permanent attachment to any particular application. The coordinator worker agent has all the information and control blocks required to complete actions within the database manager that were requested by the application.

There are four types of worker agents:

- Idle agents
- Inactive agents
- Active coordinator agents
- Subagents

Idle agents

This is the simplest form of worker agent. It does not have an outbound connection and it does not have a local database connection or an instance attachment.

Inactive agents

An inactive agent is a worker agent that is not in an active transaction, does not have an outbound connection, and does not have a local database connection or an instance attachment. Inactive agents are free to begin doing work for an application connection.

Active coordinator agents

Each process or thread of a client application has a single active agent that coordinates its work on a database. After the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using inter-process communication (IPC) or remote communication protocols. Each agent operates with its own private memory and shares database manager and database global resources such as the buffer pool with other agents. When a transaction completes, the active coordinator agent may become an inactive agent.

When a client disconnects from a database or detaches from an instance its coordinating agent will be:

- An active agent. If other connections are waiting, the worker agent becomes an active coordinator agent.

- Freed and marked as idle, if no connections are waiting and the maximum number of pool agents has not been reached.
- Terminated and its storage freed, if no connections are waiting and the maximum number of pool agents has been reached.

Subagents
In partitioned database environments and environments with intra-partition parallelism enabled, the coordinator agent distributes database requests to subagents, and these agents perform the requests for the application. After the coordinator agent is created, it handles all database requests on behalf of its application by coordinating the subagents that perform requests on the database.

Agents that are not performing work for any applications and that are waiting to be assigned are considered to be idle agents and reside in an *agent pool*. These agents are available for requests from coordinator agents operating for client programs or for subagents operating for existing coordinator agents. The number of available agents depends on the database manager configuration parameters *maxagents* and *num_poolagents*.

When an agent finishes its work but still has a connection to a database, it is placed in the agent pool. Regardless of whether the connection concentrator is enabled for the database, if an agent is not waked up to serve a new request within a certain period of time and the current number of active and pooled agents is greater than *num_poolagents*, the agent is terminated.

Agents from the agent pool (*num_poolagents*) are re-used as coordinator agents for the following kinds of applications:
- Remote TCP/IP-based applications
- Local applications on UNIX-based operating systems
- Both local and remote applications on Windows operating systems.

Other kinds of remote applications always create a new agent. If no idle agents exist when an agent is required, a new agent is created dynamically. Because creating a new agent requires a certain amount of overhead CONNECT and ATTACH performance is better if an idle agent can be activated for a client.

When a subagent is performing work for of an application, it is *associated* with that application. After it completes the assigned work, it can be placed in the agent pool, but it remains associated with the original application. When the application requests additional work, the database manager first checks the idle pool for associated agents before it creates a new agent.

Related concepts:

- "Agents in a partitioned database" on page 140
- "Configuration parameters that affect the number of agents" on page 141
- "Connection-concentrator improvements for client connections" on page 146
- "Database agent management" on page 139

DB2 architecture and process overview

General information about DB2 architecture and processes can help you understand detailed information provided for specific topics.

The following figure shows a general overview of the architecture and processes for IBM® DB2 Version 9.1.

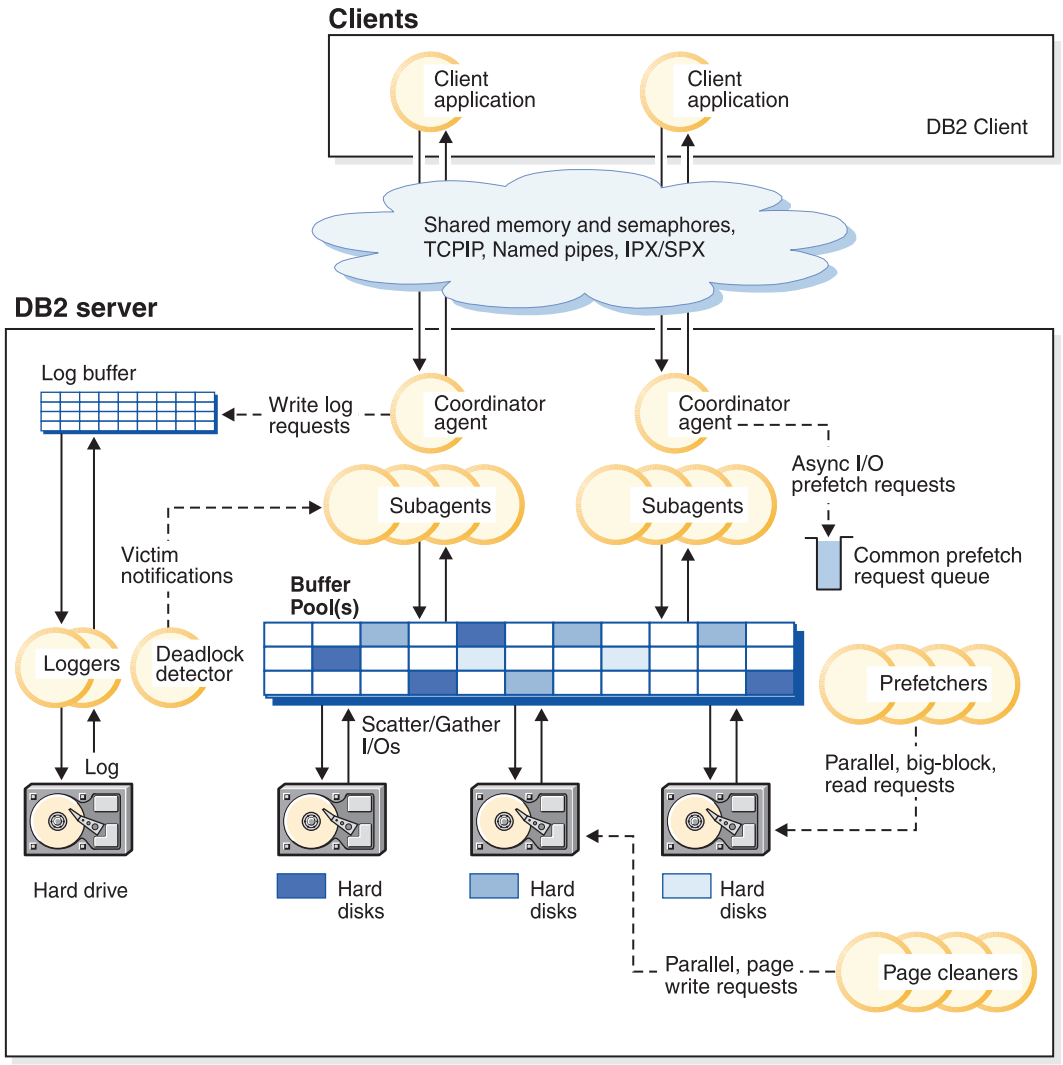


Figure 14. Architecture and Processes Overview

On the client side, either local or remote applications, or both, are linked with the DB2 client library. Local clients communicate using shared memory and semaphores; remote clients use a protocol such as Named Pipes (NPIPE), TCP/IP, NetBIOS, or SNA.

On the server side, activity is controlled by engine dispatchable units (EDUs). In all figures in this section, EDUs are shown as circles or groups of circles. EDUs are implemented as threads in a single process on Windows-based platforms and as processes on UNIX. DB2 agents are the most common type of EDUs. These agents perform most of the SQL and XQuery processing on behalf of applications. Prefetchers and page cleaners are other common EDUs.

A set of subagents might be assigned to process the client application requests. Multiple subagents can be assigned if the machine where the server resides has multiple processors or is part of a partitioned database. For example, in a symmetric multiprocessing (SMP) environment, multiple SMP subagents can exploit the many processors.

All agents and subagents are managed using a pooling algorithm that minimizes the creation and destruction of EDUs.

Buffer pools are areas of database server memory where database pages of user table data, index data, and catalog data are temporarily moved and can be modified. Buffer pools are a key determinant of database performance because data can be accessed much faster from memory than from disk. If more of the data needed by applications is present in a buffer pool, less time is required to access the data than to find it on disk.

The configuration of the buffer pools, as well as prefetcher and page cleaner EDUs, controls how quickly data can be accessed and how readily available it is to applications.

- **Prefetchers** retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the buffer pool if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter-read input operations to bring the requested pages from disk to the buffer pool. If you have multiple disks for storage of the database data, the data can be striped across the disks. Striping data lets the prefetchers use multiple disks at the same time to retrieve data.
- **Page cleaners** move data from the buffer pool back out to disk. Page cleaners are background EDUs that are independent of the application agents. They look for pages from the buffer pool that are no longer needed and write the pages to disk. Page cleaners ensure that there is room in the buffer pool for the pages being retrieved by the prefetchers.

Without the independent prefetchers and the page cleaner EDUs, the application agents would have to do all of the reading and writing of data between the buffer pool and disk storage.

Related concepts:

- "Connection-concentrator improvements for client connections" on page 146
- "Deadlocks" on page 65
- "Prefetching data into the buffer pool" on page 120
- "Database directories and files" in *Administration Guide: Planning*
- "Reducing logging overhead to improve query performance" on page 129
- "Client-server processing model" on page 142
- "Update processing" on page 119

Related reference:

- "max_connections - Maximum number of client connections " on page 424
- "max_coordagents - Maximum number of coordinating agents " on page 425

The DB2 Process Model

Knowledge of the DB2 process model can help you determine the nature of a problem because it helps you to understand how the database manager and its associated components interact.

The process model used by all DB2 servers facilitates the communication that occurs between database servers and clients and local applications. It also ensures that database applications are isolated from resources such as database control blocks and critical database files.

UNIX-based environments use an architecture based on **system processes**. For example, the DB2 communications listeners are created as system processes. Intel operating systems such as Windows use an architecture based on **threads** to maximize performance. Unless explicitly noted, this discussion uses the term ″process″ to refer to both processes and threads. You can find details of the differences between the use of Windows threads and UNIX processes later in this topic.

For each database being accessed, various processes are started to deal with the various database tasks such as prefetching, communication, and logging.

Each process of a client application has a single **coordinator agent** that operates on a database. A coordinator agent works on behalf of an application, and communicates to other agents, using interprocess communication (IPC) or remote communication protocols.

DB2 architecture provides a **firewall** so that applications run in a different address space from DB2. The firewall protects the database and the database manager from applications, stored procedures, and user-defined functions (UDFs). A firewall maintains the integrity of the data in the databases, because it disables application programming errors from overwriting internal buffers or files of the database manager. The firewall also improves reliability, because application errors cannot crash the database manager.

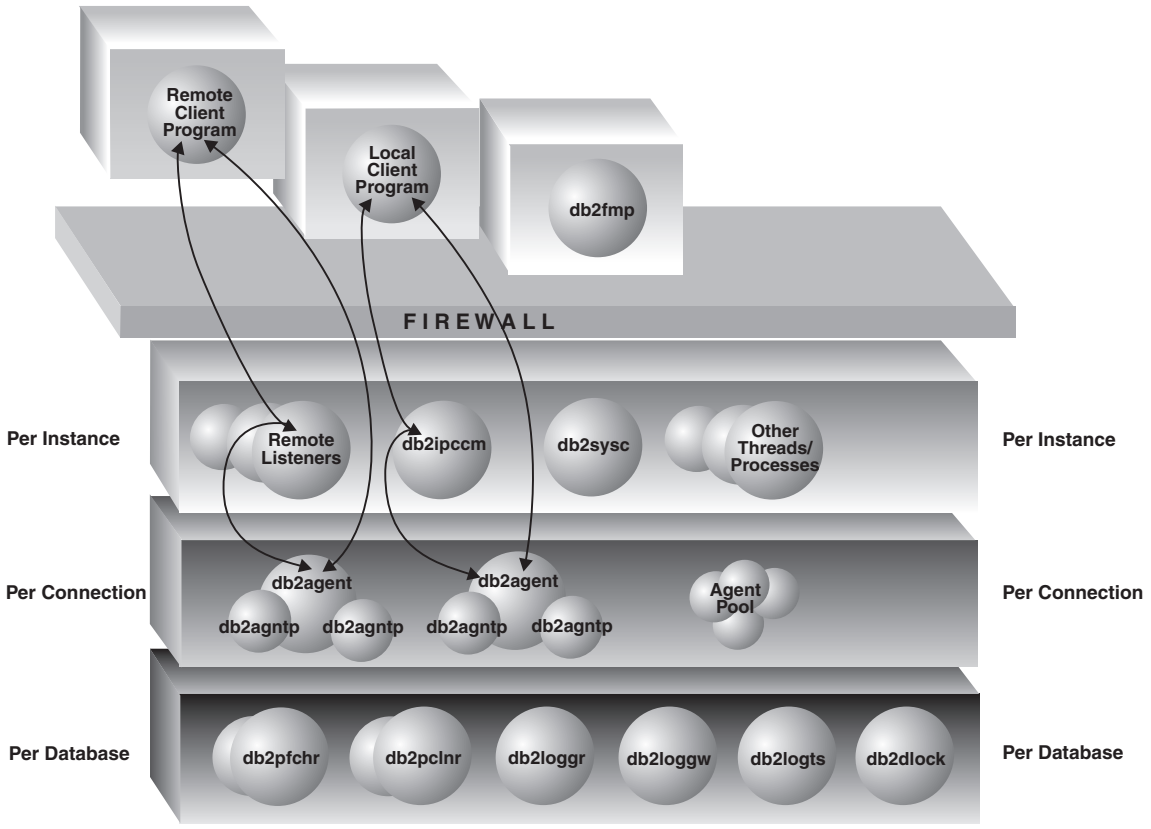


Figure 15. Process Model for DB2 Systems

The following list provides additional details on the processes shown in the figure:

Client Programs:

Client programs run remotely or on the same machine as the database server. They make their first contact with the database through a listener. A coordinator agent (db2agent) is then assigned to them.

Listeners:

Client programs make initial contact with communication listeners, which are started when DB2 is started. There is a listener for each configured communication protocol, and an interprocess communications (IPC) listener (db2ipccm) for local client programs. Listeners include:

- **db2ipccm**, for local client connections
- **db2tcpcm**, for TCP/IP connections
- **db2snacm**, for APPC connections
- **db2tcpdm**, for TCP/IP discovery tool requests

Agents:

All connection requests from client applications, whether they are local or remote, are allocated a corresponding coordinator agent (**db2agent**). When the coordinator agent is created, it performs all database requests on behalf of the application.

In some environments, in which the *intra_parallel* database manager configuration parameter is enabled, the coordinator agent distributes the database requests to subagents (**db2agntp**). These agents perform the requests for the application. Once the coordinator agent is created, it handles all database requests on behalf of its application by coordinating subagents (**db2agntp**) that perform requests on the database. Subagents that are associated with an application but are currently idle are identified by the process name **db2agnta**. Independent coordinator agents are identified by the process name **db2agnti**.

A coordinator agent may be:

- Connected to the database with an alias. For example, ″db2agent (DATA1)″ is connected to the database alias ″DATA1″.
- Attached to an instance. For example, ″db2agent (user1)″ is attached to the instance ″user1″.

An additional type of agent, **db2agnsc** is created by DB2 to perform certain operations in parallel. In particular, they are used in database recovery actions.

Idle agents reside in an agent pool. These agents are available for requests from coordinator agents operating on behalf of client programs, or from subagents operating on behalf of existing coordinator agents. The number of available agents is dependent on the database manager configuration parameters *maxagents* and *num_poolagents*.

db2fmp:

The fenced mode process. It is responsible for executing fenced stored procedures and user-defined functions outside the firewall. db2fmp is always a separate process but may be multithreaded depending on the types of routines it executes.

Database Threads/Processes:

The following list includes some of the important threads/processes used by each database:

- **db2pfchr**, for buffer pool prefetchers
- **db2pclnr**, for buffer pool page cleaners
- **db2loggr**, for manipulating log files to handle transaction processing and recovery
- **db2loggw**, for writing log records to the log files.
- **db2logts**, for collecting historical information about which logs are active when a tablespace is modified. This information is ultimately recorded in the DB2TSCHG.HIS file in the database directory. It is used to speed up tablespace rollforward recovery.
- **db2dlock**, for deadlock detection. In a multi-partitioned database environment, an additional process called **db2glock** is used to coordinate the information gathered from the db2dlock process on each partition. db2glock runs only on the catalog partition.
- **db2taskd**, for distribution of background database tasks. The tasks are executed by processes called **db2taskp**.
- **db2hadrp**, HADR primary server process
- **db2hadrs**, HADR standby server process
- **db2lfr**, for log file readers that processes individual log files
- **db2logts** tracks which table spaces have log records in each log file

- **db2shred** processes individual log records within log pages
- **db2redom**, for the redo master. During recovery, processes redo log records and assigns log records to redo workers for processing.
- **db2redow**, for the redo worker. During recovery, processes redo log records at the request of the redo master.
- **db2logmgr**, for the log manager. Manages log files for a recoverable database
- event monitor processes are identified as follows:
  - **db2evm%1%2 (%3)** where **%1** can be
    - **g**- global file event monitor
    - **l** - local file event monitor
    - **t** - table event monitor
    - **gp** - global piped event monitor
    - **lp** - local piped event monitor

    where **%2** can be
    - **i** - coordinator
    - **p** - not coordinator

    and **%3** is the event monitor name
- backup and restore processes are identified as follows:
  - **db2bm.%1.%2** backup and restore buffer manipulator, and **db2med.%1.%2**, backup and restore media controller, where%
    - **%1**- the process number of the agent that controls the backup or restore session
    - **%2**- a sequential value used to disambiguate among (possibly many) processes belonging to a particular backup or restore session

    For example: `db2bm.13579.2` identifies the second db2bm process that is controlled by the db2agent process with pid 13579.

Database Server Threads and Processes:

The system controller (**db2sysc**) must exist in order for the database server to function. Also, the following threads and processes may be started to carry out various tasks:

- **db2resync**, the resync agent that scans the global resync list
- **db2gds**, the global daemon spawner on UNIX-based systems that starts new processes
- **db2wdog**, the watchdog on UNIX-based systems that handles abnormal terminations
- **db2fcms**, the fast communications manager sender daemon
- **db2fcmr**, the fast communications manager receiver daemon
- **db2pdbc**, the parallel system controller, which handles parallel requests from remote nodes (used only in a partitioned database environment).
- **db2cart**, for archiving log files when accessing a database configured with USEREXIT enabled
- **db2fmtlg**, for formatting log files, when accessing a database configured with LOGRETAIN enabled, but with USEREXIT disabled
- **db2panic**, the panic agent, which handles urgent requests after agent limits have been reached at a particular node (used only in a partitioned database environment)

- **db2srvlst**, manages lists of addresses for systems such as DB2 for z/OS.
- **db2fmd**, the fault monitor daemon
- **db2disp**, the client connection concentrator dispatcher
- **db2acd**, autonomic computing daemon hosting the health monitor and automatic maintenance utilities. This process was formerly called **db2hmon**.

Differences between Windows and UNIX

DB2 for Windows differs from UNIX-based environments in that the database engine is multi-threaded, not multi-processed. In Windows systems, each of the dispatchable units on the agent side of the firewall is a thread under the process db2sysc. This allows the database engine to let the operating system perform task-switching at the thread level instead of the process level. For each database being accessed, threads are started to deal with database tasks (for example, prefetching).

Another difference is in the handling of abnormal terminations. There is no need for a "watchdog" process in Windows systems, because these systems ensure that the allocated resources are cleaned up after an abnormal termination. Thus, there is no equivalent of the db2wdog process on the Windows systems. In addition, the db2gds process or thread is not needed on the Windows systems, which have their own mechanisms for starting threads.

Related concepts:

- "First failure data capture information" in *Troubleshooting Guide*
- "Determining active process status" in *Troubleshooting Guide*

Database agent management

Most applications establish a one-to-one relationship between the number of connected applications and the number of application requests that can be processed by the database. However, it may be that your work environment is such that you require a many-to-one relationship between the number of connected applications and the number of application requests that can be processed.

The ability to control these factors separately is provided by two database manager configuration parameters:

- The *max_connections* parameter, which specifies the number of connected applications
- The *max_coordagents* parameter, which specifies the number of application requests that can be processed

The connection concentrator is enabled when the value of *max_connections* is greater than the value of *max_coordagents*.

Because each active coordinator agents requires global resource overhead, the greater the number of these agents the greater the chance that the upper limits of available database global resources will be reached. To prevent reaching the upper limits of available database global resources, you might set the value of *max_connections* higher than the value of *max_coordagents*.

Related concepts:

- "Agents in a partitioned database" on page 140
- "Configuration parameters that affect the number of agents" on page 141

- "Connection-concentrator improvements for client connections" on page 146

Agents in a partitioned database

For partitioned database environments and environments with intra-partition parallelism enabled, each database partition (that is, each database server or node) has its own pool of agents from which subagents are drawn. Because of this pool, subagents do not have to be created and destroyed each time one is needed or is finished its work. The subagents can remain as associated agents in the pool and be used by the database manager for new requests from the application they are associated with.

Note: If the connection concentrator is enabled, subagents are not necessarily associated with an application.

For partitioned database environments and environments with intra-partition parallelism enabled, the impact to performance and memory costs within the system is strongly related to how your agent pool is tuned:

- The database manager configuration parameter for agent pool size (*num_poolagents*) affects the total number of subagents that can be kept associated with applications on a database partition, which is also called a node. If the pool size is too small and the pool is full, a subagent disassociates itself from the application it is working on and terminates. Because subagents must be constantly created and re-associated to applications, performance suffers.

  In addition, if the value of *num_poolagents* is too small, one application may fill the pool with associated subagents. Then when another application requires a new subagent and has no subagents in its associated agent pool, it will recycle inactive subagents from the agent pools of other applications. This behavior ensures that resources are fully utilized.
- Weigh concerns about having too few agents against the resource costs of allowing too many agents to be active at any given time.

  For example, if the value of *num_poolagents* is too large, associated subagents may sit unused in the pool for long periods of time, using database manager resources that are not available for other tasks.

  Note: When the connection concentrator is enabled, the number of agents specified by *num_poolagents* is only advisory. More agents might be in the agent pool at any given time.

Other asynchronous processes and threads

In addition to the database agents, other asynchronous database-manager activities run as their own process or thread including:

- Database I/O servers or I/O prefetchers
- Database asynchronous page cleaners
- Database loggers
- Database deadlock detectors
- Event monitors
- Communication and IPC listeners
- Table space container rebalancers.

Related concepts:

- "Configuration parameters that affect the number of agents" on page 141
- "Database agents" on page 131
- "Database agent management" on page 139
- "I/O server configuration for prefetching and parallelism" on page 124
- "Illustration of prefetching with parallel I/O" on page 125

Configuration parameters that affect the number of agents

The following database manager configuration parameters determine how many database agents are created and how they are managed:

- Maximum Number of Agents (*maxagents*): The number of agents that can be working at any one time. This value applies to the total number of agents that are working on all applications, including coordinator agents, subagents, inactive agents, and idle agents.
- Agent Pool Size (*num_poolagents*): The total number of agents, including active agents and agents in the agent pool, that are kept available in the system. The default value for this parameter is half the number specified for *maxagents*.
- Initial Number of Agents in Pool (*num_initagents*): When the database manager is started, a pool of worker agents is created based on this value. This speeds up performance for initial queries. The worker agents all begin as idle agents.
- Maximum Number of Connections (*max_connections*): specifies the maximum number of connections allowed to the database manager system on each database partition.
- Maximum Number of Coordinating Agents (*max_coordagents*): For partitioned database environments and environments with intra-partition parallelism enabled when the Connection Concentrator is enabled, this value limits the number of coordinating agents.
- Maximum Number of Concurrent Agents (*maxcagents*): This value controls the number of *tokens* permitted by the database manager. For each database transaction (unit of work) that occurs when a client is connected to a database, a coordinating agent must obtain permission to process the transaction from the database manager. This permission is called a *processing token*. The database manager permits only agents that have a processing token to execute a unit of work against a database. If a token is not available, the agent must wait until one is available to process the transaction.

  This parameter can be useful in an environment in which peak usage requirements exceed system resources for memory, CPU, and disk. For example, in such an environment, paging might cause performance degradation for peak load periods. You can use this parameter to control the load and avoid performance degradation, although it can affect either concurrency or wait time, or both.

Related concepts:

- "Agents in a partitioned database" on page 140
- "Database agents" on page 131
- "Database agent management" on page 139
- "Connection concentrator" in *DB2 Connect User's Guide*

Client-server processing model

Local and remote application processes can work with the same database. A remote application is one that initiates a database action from a machine that is remote from the database machine. Local applications are directly attached to the database at the server machine.

How DB2 manages client connections depends on whether the connection concentrator is on or off. The connection concentrator is ON when the *max_connections* database manager configuration parameter is set larger than the *max_coordagents* configuration parameter.

- If the connection concentrator is OFF, each client application is assigned a unique EDU called a *coordinator agent* that coordinates the processing for that application and communicates with it.
- If the connection concentrator is ON, each coordinator agent can manage many client connections, one at a time, and might coordinate the other worker agents to do this work. For Internet applications with many relatively transient connections, or similar applications with many relatively small transactions, the connection concentrator improves performance by allowing many more client applications to be connected. It also reduces system resource use for each connection.

Each of the circles of the following figure represent engine dispatchable units (EDUs) which are known as “processes” on UNIX platforms, and “threads” on Windows Operating systems.

A means of communicating between an application and the database manager must be established before the work the application wants done at the database can be carried out.

At A1 in the figure below, a local client establishes communications first through the db2ipccm. At A2, the db2ipccm works with a db2agent EDU, which becomes the coordinator agent for the application requests from the local client. The coordinator agent then contacts the client application at A3 to establish shared memory communications between the client application and the coordinator. The application at the local client is connected to the database at A4.

At B1 in the figure below, a remote client establishes communications through the db2tcpcm EDU. If any other communications protocol is chosen, the appropriate communication manager is used. The db2tcpcm EDU establishes TCP/IP communication between the client application and the db2tcpcm. It then works with a db2agent at B2, which becomes the coordinator agent for the application and passes the connection to this agent. At B3 the coordinator agent contacts the remote client application and is connected to the database.

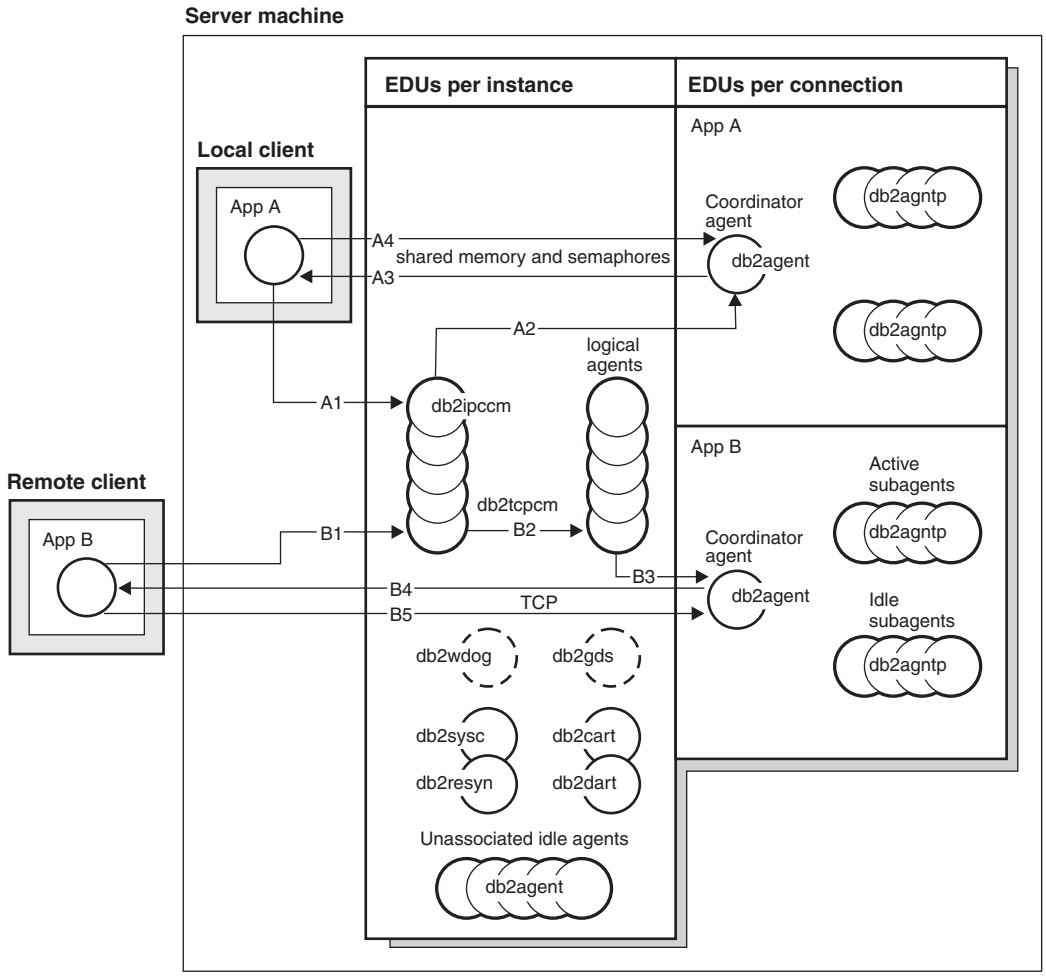


Figure 16. Process model overview

Other things to notice in this figure:

- Worker agents carry out application requests.
- There are four types of worker agents: active coordinator agents, active subagents, associated subagents, and idle agents.
- Each client connection is linked to an active coordinator agent.
- In a partitioned database environment, and enabled intra-partition parallelism environments, the coordinator agents distribute database requests to subagents (db2agntp). The subagents perform the requests for the application.
- There is an agent pool (db2agent) where idle and pooled agents wait for new work.
- Other EDUs manage client connections, logs, two-phase COMMITs, backup and restore tasks, and other tasks.

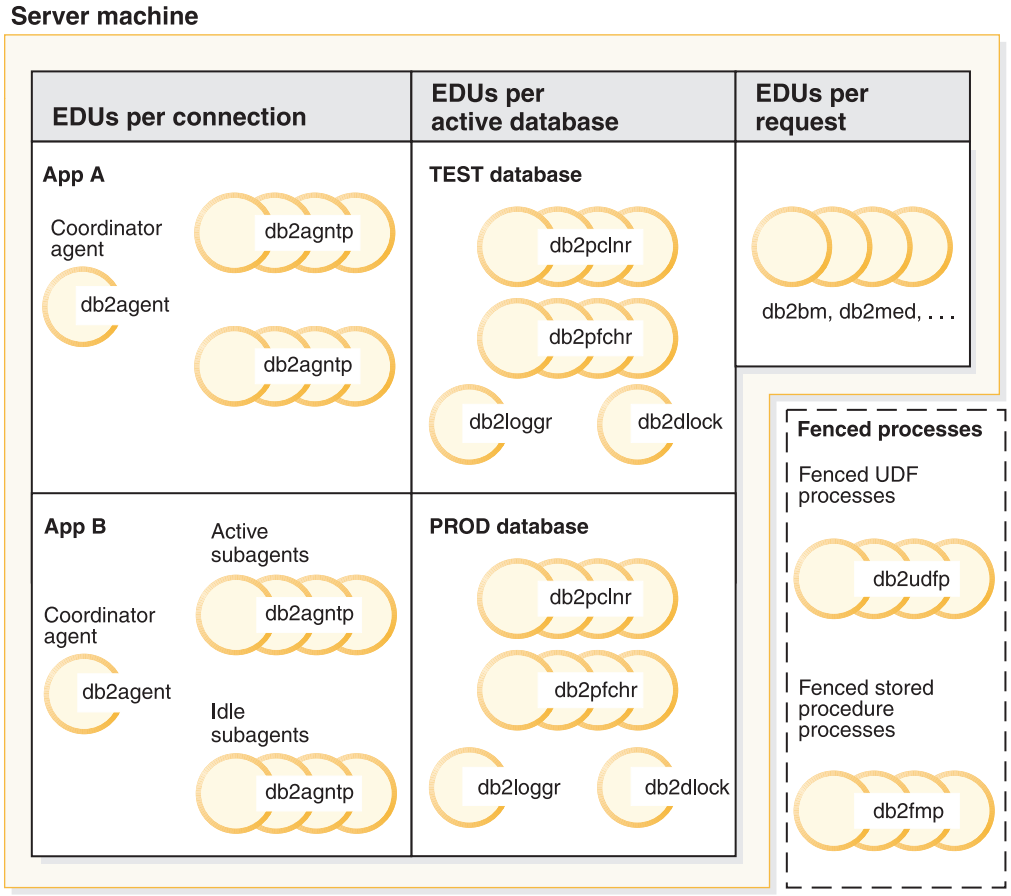


Figure 17. Process model, part 2

This figure shows additional engine dispatchable units (EDUs) that are part of the server machine environment. Each active database has its own shared pool of prefetchers (db2pfchr) and page cleaners (db2pclnr), and its own logger (db2loggr) and deadlock detector (db2dlock).

Fenced user-defined functions (UDFs) and stored procedures, which are not shown in the figure, are managed to minimize costs associated with their creation and destruction. The default for the *keepfenced* database manager configuration parameter is "YES", which keeps the stored procedure process available for re-use at the next stored procedure call.

Note: Unfenced UDFs and stored procedures run directly in an agent's address space for better performance. However, because they have unrestricted access to the agent's address space, they need to be rigorously tested before being used.

The multiple database partition processing model is a logical extension of the single database partition processing model. In fact, a single common code base supports both modes of operation. The following figure shows the similarities and differences between the single database partition processing model as seen in the previous two figures, and the multiple database partition processing model.

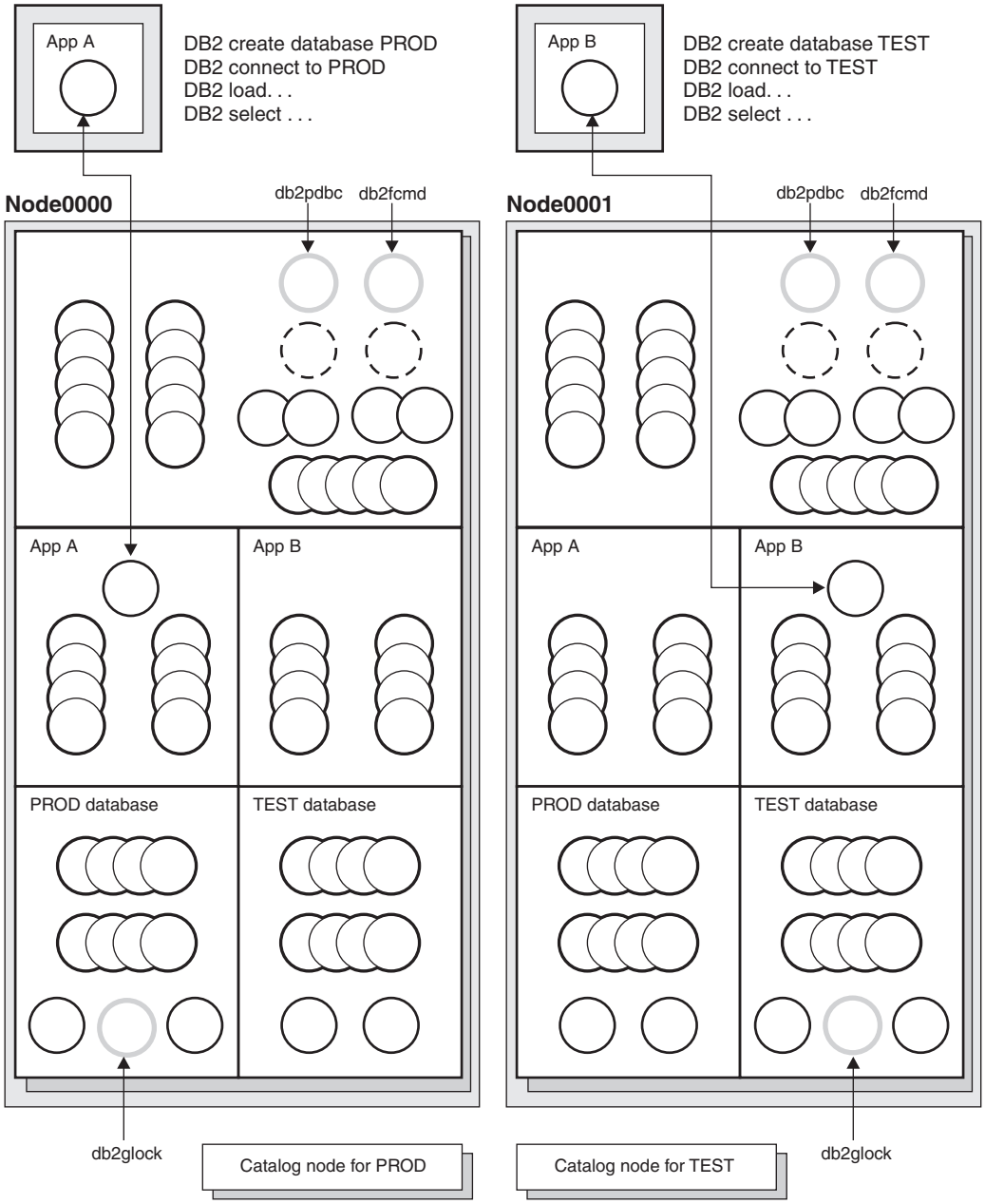


Figure 18. Process model and multiple database partitions

Most engine dispatchable units (EDUs) are the same between the single database partition processing model and the multiple database partition processing model.

In a multiple database partition (or node) environment, one of the database partitions is the catalog node. The catalog keeps all of the information relating to the objects in the database.

As shown in the figure above, because Application A creates the PROD database on Node0000, the catalog for the PROD database is created on this node. Similarly, because Application B creates the TEST database on Node0001, the catalog for the TEST database is created on this node. You might want to create your databases on different nodes to balance the extra activity associated with the catalogs for each database across the nodes in your system environment.

There are additional EDUs (db2pdbc and db2fcmd) associated with the instance and these are found on each node in a multiple partition database environment. These EDUs are needed to coordinate requests across database partitions and to enable the Fast Communication Manager (FCM).

There is also an additional EDU (db2glock) associated with the catalog node for the database. This EDU controls global deadlocks across the nodes where the active database is located.

Each CONNECT from an application is represented by a connection that is associated with a coordinator agent to handle the connection. The *coordinator agent* is the agent that communicates with the application, receiving requests and sending replies. It can either satisfy the request itself or coordinate multiple subagents to work on the request. The database partition where the coordinator agent exists is called the *coordinator node* of that application. The coordinator node can also be set with the SET CLIENT CONNECT_NODE command.

Parts of the database requests from the application are sent by the coordinator node to subagents at the other database partitions; and all results from the other database partitions are consolidated at the coordinator node before being sent back to the application.

The database partition where the CREATE DATABASE command was issued is called the "catalog node" for the database. It is at this database partition that the catalog tables are stored. Typically, all user tables are distributed across a set of nodes.

Note: Any number of database partitions can be configured to run on the same machine. This is known as a "multiple logical partition", or "multiple logical node", configuration. Such a configuration is very useful on large symmetric multiprocessor (SMP) machines with very large main memory. In this environment, communications between database partitions can be optimized to use shared memory and semaphores.

Related concepts:

- "DB2 architecture and process overview" on page 132
- "Connection-concentrator improvements for client connections" on page 146
- "Reducing logging overhead to improve query performance" on page 129
- "Update processing" on page 119

Connection-concentrator improvements for client connections

For Internet applications with many relatively transient connections, or similar kinds of applications, the connection concentrator improves performance by allowing many more client connections to be processed efficiently. It also reduces memory use for each connection and decreases the number of context switches.

Note: The connection concentrator is enabled when the value of *max_connections* is greater than the value of *max_coordagents*.

In an environment that requires many simultaneous user connections, you can enable the connection concentrator for more efficient use of system resources. This feature incorporates advantages formerly found only in DB2 Connect™ connection pooling. Both connection pooling and the connection concentrator are described in

the DB2 Connect User's Guide. After the first connection, the connection concentrator reduces the connect time to a host. When a disconnection from a host is requested, the inbound connection is dropped, but the outbound connection to the host is kept in a pool. When a new request is made to connect to the host, DB2 tries to reuse an existing outbound connection from the pool.

Note: When applications use connection pooling or the connection concentrator, for best performance tune the parameters that control the size of the block of data that is cached. For more information, refer to the DB2 Connect User's Guide.

With DB2Connect connection pooling and the connection concentrator, the active agent does not close its outbound connection after a client disconnects, but is placed in the agent pool for the application, where it becomes a *logical subagent*, which is controlled by a *logical coordinator agent*. with an active connection to the remote host.

When using connection pooling, DB2 Connect is restricted to inbound TCP/IP and to outbound TCP/IP and SNA connections. When working with SNA, the security type must be `NONE` for the connection to be placed in the pool. In connection pooling these idle agents are called *inactive agents*. The pool of inactive agents is a synonym for the outbound connection pool. The connection concentrator implements a similar method of retaining inactive agents for later use in an application-specific pool.

Usage examples:

1. Consider an ESE environment with a single database partition in which, on average, 1000 users are connected to the database. At times, the number of concurrent transactions is as high as 200, but never higher than 250. Transactions are short.

   For this workload, the administrator sets the following database manager configuration parameters:

   - *max_connections* is set to 1000 to ensure support for the average number of connections.
   - *max_coordagents* is set to 250 to support the maximum number of concurrent transactions.
   - *maxagents* is set high enough to support all of the coordinator agents and subagents (where applicable) that are required to execute transactions on the node.

     If *intra_parallel* is OFF, *maxagents* is set to 250 because in such an environment, there are no subagents. If *intra_parallel* is ON, *maxagents* should be set large enough to accommodate the coordinator agent and the subagents required for each transaction that accesses data on the node. For example, if each transaction requires 4 subagents, *maxagents* should be set to (4+1) * 250, which is 1250. To tune *maxagents* further, take monitor snapshots for the database manager. The high-water mark of the agents will indicate the appropriate setting for *maxagents*.
   - *num_poolagents* is set to at least 250, or as high as 1250, depending on the value of *maxagents* to ensure that enough database agents are available to service incoming client requests without the overhead of creating new ones.

     However, this number could be lowered to reduce resource usage during low-usage periods. Setting this value too low causes agents to be deallocated instead of going into the agent pool, which requires new agents to be created before the server is able to handle an average workload.

- *num_initagents* is set to be the same as *num_poolagents* because you know the number of agents that should be active. This causes the database to create the appropriate number of agents when it starts instead of creating them before a given request can be handled.

The ability of the underlying hardware to handle a given workload is not discussed here. If the underlying hardware cannot handle X-number of agents working at the same time, then you need to reduce this number to the maximum that the hardware can support. For example, if the maximum is only 1500 agents, then this limits the maximum number of concurrent transactions that can be handled. You should monitor this kind of performance-related setting because it is not always possible to determine exact requests sent to other nodes at at given point in time.

2. In a system in which the workload needs to be restricted to a maximum 100 concurrent transactions and the same number of connected users as in example 1, you can set database manager configuration parameters as follows:
   - *max_coordagents* is set to 100
   - *num_poolagents* is set to 100

   With these settings, the maximum number of clients that can concurrently execute transactions is 100. When all clients disconnect, 100 agents are waiting to service new client connections. However, you should set *maxagents*, based on the type of workload, intra-query parallelism settings, the number of database partitions, and the underlying hardware.
3. Consider next an ESE installation with five database partitions, in which each database partition has an average of 1000 user connections, and the concurrent transactions are as high as 200 but never higher than 250, set database configuration parameters as follows:
   - *max_coordagents* is set to 250 because, as in example 1, at most 250 clients execute transactions concurrently.
   - *maxagents* is set to 1500 Assuming data is distributed across the five database partitions, each transaction may execute at least one subsection on each of the nodes in the system. ((1 coordinator agent + 5 subagents) * 250 = 1500.)
   - *num_poolagents* is set to 1200. (Assuming an average of 200 concurrent transaction with a maximum of 250. As a result, the number of agents required on average will be (1+5)*200 = 1200.)
   - *num_initagents* is set to be the same as *num_poolagents*, as in example 1.
4. In a system for which you do not want to enable the connection concentrator but want to allow for 250 connected users at one time, set the database manager configuration parameters as follows:
   - *max_connections* is set to 250.
   - *max_coordagents* is set to 250.

Related concepts:

- "Database agents" on page 131
- "DB2 architecture and process overview" on page 132
- "Database agent management" on page 139

Chapter 9. Designing for Optimal Data Access

The SQL and XQuery compiler process

The SQL and XQuery compiler performs several steps to produce an access plan that can be executed. These steps are shown in the following figure and described in the sections below the figure. Note that some steps occur only for queries in a federated database.

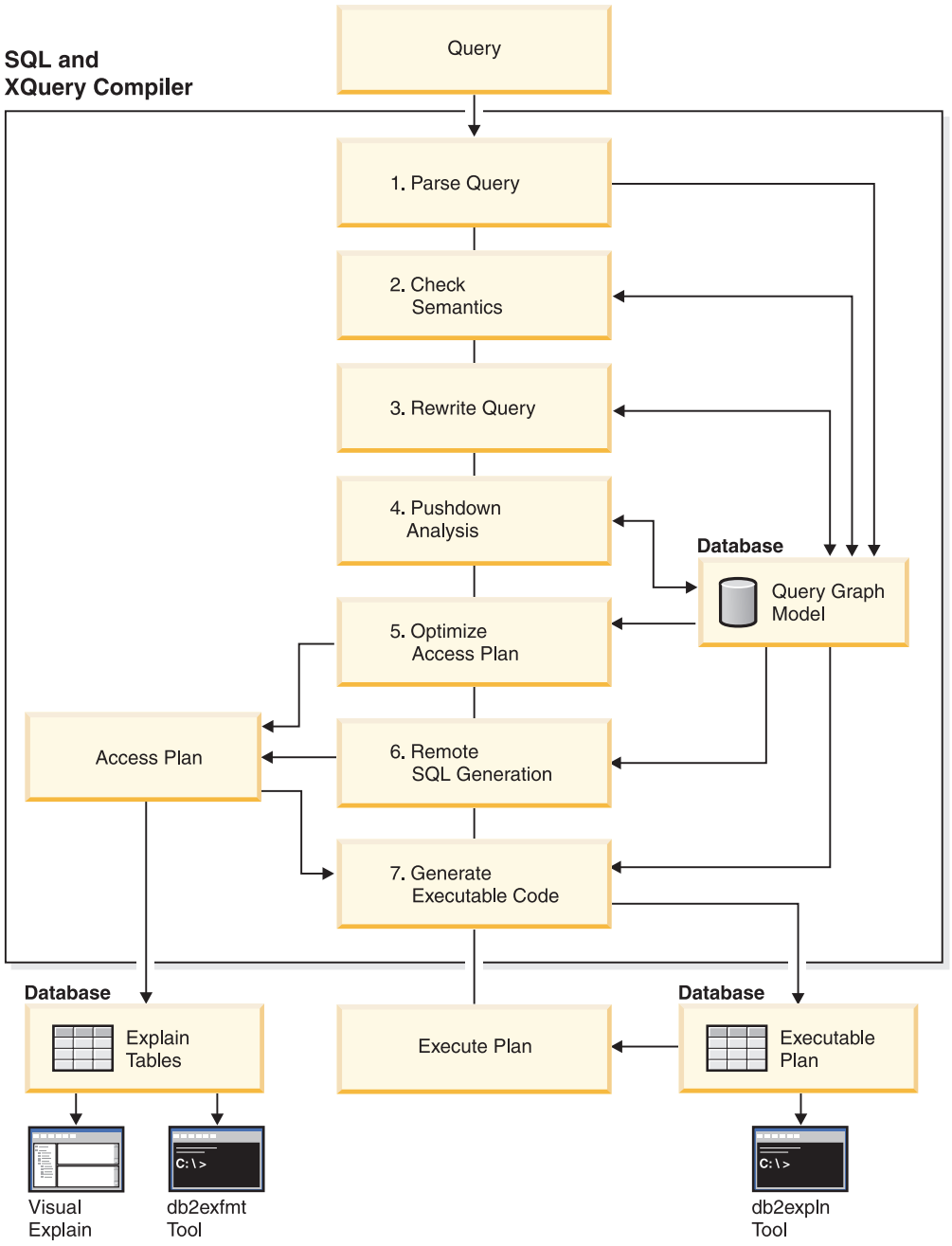


Figure 19. Steps performed by SQL and XQuery compiler

Query Graph Model

The *query graph model* is an internal, in-memory database that represents the query as it is processed in the steps described below:

1. **Parse Query**

   The SQL and XQuery compiler analyzes the query to validate the syntax. If any syntax errors are detected, the query compiler stops processing and returns the appropriate error to the application that submitted the query. When parsing is complete, an internal representation of the query is created and stored in the query graph model.

2. **Check Semantics**

The compiler ensures that there are no inconsistencies among parts of the statement. As a simple example of semantic checking, the compiler verifies that the data type of the column specified for the YEAR scalar function is a datetime data type.

The compiler also adds the behavioral semantics to the query graph model, including the effects of referential constraints, table check constraints, triggers, and views. The query graph model contains all of the semantics of the query, including query blocks, subqueries, correlations, derived tables, expressions, data types, data type conversions, code page conversions, and distribution keys.

3. **Rewrite Query**

   The compiler uses the global semantics stored in the query graph model to transform the query into a form that can be optimized more easily and stores the result in the query graph model.

   For example, the compiler might move a predicate, altering the level at which it is applied and potentially improving query performance. This type of operation movement is called *general predicate pushdown*. In a partitioned database environment, the following query operations are more computationally intensive:

   - Aggregation
   - Redistribution of rows
   - Correlated subqueries, which are subqueries that contain a reference to a column of a table that is outside of the subquery.

   For some queries in a partitioned database environment, decorrelation might occur as part of rewriting the query.

4. **Pushdown Analysis (Federated Databases)**

   The major task in this step is to recommend to the optimizer whether an operation can be remotely evaluated or *pushed-down* at a data source. This type of pushdown activity is specific to data source queries and represents an extension to general predicate pushdown operations.

   This step is bypassed unless you are executing federated database queries.

5. **Optimize Access Plan**

   Using the query graph model as input, the optimizer portion of the compiler generates many alternative execution plans for satisfying the query. To estimate the execution cost of each alternative plan, the optimizer uses the statistics for tables, indexes, columns and functions. Then it chooses the plan with the smallest estimated execution cost. The optimizer uses the query graph model to analyze the query semantics and to obtain information about a wide variety of factors, including indexes, base tables, derived tables, subqueries, correlations and recursion.

   The optimizer can also consider another type of pushdown operation, *aggregation and sort*, which can improve performance by pushing the evaluation of these operations to the Data Management Services component.

   The optimizer also considers whether there are different sized buffer pools when determining page size selection. That the environment includes a partitioned database is also considered as well as the ability to enhance the chosen plan for the possibility of intra-query parallelism in a symmetric multi-processor (SMP) environment. This information is used by the optimizer to help select the best access plan for the query.

   The output of this step of the compiler is an access plan. This access plan provides the information captured in the Explain tables. The information used to generate the access plan can be captured with an explain snapshot.

6. **Remote SQL Generation (Federated Databases)**

   The final plan selected by the optimizer might consist of a set of steps that operate on a remote data source. For operations that are performed by each data source, the remote SQL generation step creates an efficient SQL statement based on the data-source SQL dialect.

7. **Generate "Executable" Code**

   In the final step, the compiler uses the access plan and the query graph model to create an executable access plan, or section, for the query. This code generation step uses information from the query graph model to avoid repetitive execution of expressions that need to be computed only once for a query. Examples for which this optimization is possible include code page conversions and the use of host variables.

   To enable query (re)optimization of static and dynamic SQL and XQuery statements that have host variables, special registers, or parameter markers, bind the package with the REOPT bind option. If used, the access path for an SQL or XQuery statement, belonging to that package and containing host variables, parameter markers or special registers, will be optimized using the values of these variables rather than default estimates chosen by the compiler. This optimization takes place at query execution time when the values are available.

   Information about access plans for static SQL and XQuery statements is stored in the system catalog tables. When the package is executed, the database manager will use the information stored in the system catalog tables to determine how to access the data and provide results for the query. This information is used by the *db2expln* tool.

Note: Execute RUNSTATS at appropriate intervals on tables that change often. The optimizer needs up-to-date statistical information about the tables and their data to create the most efficient access plans. Rebind your application to take advantage of updated statistics. If RUNSTATS is not executed or the optimizer suspects that RUNSTATS was executed on empty or nearly empty tables, it may either use defaults or attempt to derive certain statistics based on the number of file pages used to store the table on disk (FPAGES). The total number of occupied blocks is stored in the ACTIVE_BLOCKS column.

Related concepts:

- "Advantages of deferred binding" in *Developing Embedded SQL Applications*
- "Materialized query tables" on page 11
- "Data-access methods" on page 164
- "Effects of sorting and grouping" on page 172
- "Guidelines for analyzing where a federated query is evaluated" on page 214
- "Joins" on page 187
- "Optimization strategies for intra-partition parallelism" on page 177
- "Optimization strategies for MDC tables" on page 18
- "Predicate typology and access plans" on page 170
- "Query rewriting methods and examples" on page 203

Related reference:

- "Restrictions on native XML data store" in *XML Guide*

Choosing an optimization class

Setting the optimization class can provide some of the advantages of explicitly specifying optimization techniques, particularly for the following reasons:

- To manage very small databases or very simple dynamic queries
- To accommodate memory limitations at compile time on your database server
- To reduce the query compilation time, such as PREPARE.

Most statements can be adequately optimized with a reasonable amount of resources by using optimization class 5, which is the default query optimization class. At a given optimization class, the query compilation time and resource consumption is primarily influenced by the complexity of the query, particularly the number of joins and subqueries. However, compilation time and resource usage are also affected by the amount of optimization performed.

Query optimization classes 1, 2, 3, 5, and 7 are all suitable for general-purpose use. Consider class 0 only if you require further reductions in query compilation time and you know that the SQL and XQuery statements are extremely simple.

Tip: To analyze queries that run a long time, run the query with db2batch to find out how much time is spent in compilation and how much is spent in execution.If compilation requires more time, reduce the optimization class. If execution requires more time, consider a higher optimization class.

When you select an optimization class, consider the following general guidelines:

- Start by using the default query optimization class, class 5.
- To use a class other than the default, try class 1, 2 or 3 first. Classes 0, 1, and 2 use the Greedy join enumeration algorithm.
- Use optimization class 1 or 2 if you have many tables with many of the join predicates that are on the same column, and if compilation time is a concern.
- Use a low optimization class (0 or 1) for queries having very short run-times of less than one second. Such queries tend to have the following characteristics:
  - Access to a single or only a few tables
  - Fetch a single or only a few rows
  - Use fully qualified, unique indexes.

Online transaction processing (OLTP) transactions are good examples of this kind of query.

- Use a higher optimization class (3, 5, or 7) for longer running queries that take more than 30 seconds.
- Classes 3 and above use the Dynamic Programming join enumeration algorithm. This algorithm considers many more alternative plans, and might incur significantly more compilation time than classes 0, 1, and 2, especially as the number of tables increases.
- Use optimization class 9 only if you have specific extraordinary optimization requirements for a query.

Complex queries might require different amounts of optimization to select the best access plan. Consider using higher optimization classes for queries that have the following characteristics:

- Access to large tables
- A large number of predicates
- Many subqueries

- Many joins
- Many set operators, such as UNION and INTERSECT
- Many qualifying rows
- GROUP BY and HAVING operations
- Nested table expressions
- A large number of views.

Decision support queries or month-end reporting queries against fully normalized databases are good examples of complex queries for which at least the default query optimization class should be used.

Use higher query optimization classes for SQL and XQuery statements that were produced by a query generator. Many query generators create inefficient queries. Poorly written queries, including those produced by a query generator, require additional optimization to select a good access plan. Using query optimization class 2 and higher can improve such SQL and XQuery queries.

Note: In a federated database query, the optimization class does not apply to the remote optimizer.

Related concepts:

- "Benchmark testing" on page 357
- "Configuration parameters that affect query optimization" on page 158
- "Optimization strategies for intra-partition parallelism" on page 177
- "Optimization strategies for MDC tables" on page 18

Related tasks:

- "Setting the optimization class" on page 157

Related reference:

- "Optimization classes" on page 154

Optimization classes

When you compile an SQL or XQuery query, you can specify an optimization class that determines how the optimizer chooses the most efficient access plan for that query. The optimization classes are differentiated by the number and type of optimization strategies considered in the compilation of the query. Although you can specify optimization techniques individually to improve runtime performance for the query, the more optimization techniques you specify, the more time and system resources query compilation will require.

You can specify one of the following optimizer classes when you compile an SQL or XQuery query:

0 - This class directs the optimizer to use minimal optimization to generate an access plan. This optimization class has the following characteristics:
- Non-uniform distribution statistics are not considered by the optimizer.
- Only basic query rewrite rules are applied.
- Greedy join enumeration occurs.
- Only nested loop join and index scan access methods are enabled.
- List prefetch is not used in generated access methods.
- The star-join strategy is not considered.

This class should only be used in circumstances that require the the lowest possible query compilation overhead. Query optimization class 0 is appropriate for an application that consists entirely of very simple dynamic SQL or XQuery statements that access well-indexed tables.

1 - This optimization class has the following characteristics:

- Non-uniform distribution statistics are not considered by the optimizer.
- Only a subset of the query rewrite rules are applied.
- Greedy join enumeration occurs.
- List prefetch is not used in generated access methods.

Optimization class 1 is similar to class 0 except that Merge Scan joins and table scans are also available.

2 - This class directs the optimizer to use a degree of optimization significantly higher than class 1, while keeping the compilation cost significantly lower than classes 3 and above for complex queries. This optimization class has the following characteristics:

- All available statistics, including both frequency and quantile non-uniform distribution statistics, are used.
- All query rewrite rules are applied, including routing queries to materialized query tables, except computationally intensive rules that are applicable only in very rare cases.
- Greedy join enumeration is used.
- A wide range of access methods are considered, including list prefetch and materialized query table routing.
- The star-join strategy is considered, if applicable.

Optimization class 2 is similar to class 5 except that it uses Greedy join enumeration instead of Dynamic Programming. This class has the most optimization of all classes that use the Greedy join enumeration algorithm, which considers fewer alternatives for complex queries, and therefore consumes less compilation time than classes 3 and above. Class 2 is recommended for very complex queries in a decision support or online analytic processing (OLAP) environment. In such environments, specific queries are rarely repeated exactly, so that a query access plan is unlikely to remain in the cache until the next occurrence of the query.

3 - This class requests a moderate amount of optimization. This class comes closest to matching the query optimization characteristics of DB2 for MVS/ESA, OS/390®, or z/OS. This optimization class has the following characteristics:

- Non-uniform distribution statistics, which track frequently occurring values, are used if available.
- Most query rewrite rules are applied, including subquery-to-join transformations.
- Dynamic programming join enumeration, as follows:
  - Limited use of composite inner tables
  - Limited use of Cartesian products for star schemas involving look-up tables
- A wide range of access methods are considered, including list prefetch, index ANDing, and star joins.

This class is suitable for a broad range of applications. This class improves access plans for queries with four or more joins. However, the optimizer might fail to consider a better plan that might be chosen with the default optimization class.

5 - This class directs the optimizer to use a significant amount of optimization to generate an access plan. This optimization class has the following characteristics:

- All available statistics are used, including both frequency and quantile distribution statistics.
- All of the query rewrite rules are applied, including the routing of queries to materialized query tables, except for those computationally intensive rules which are applicable only in very rare cases.
- Dynamic programming join enumeration, as follows:
  - Limited use of composite inner tables
  - Limited use of Cartesian products for star schemas involving look-up tables
- A wide range of access methods are considered, including list prefetch, index ANDing, and materialized query table routing.

When the optimizer detects that the additional resources and processing time are not warranted for complex dynamic SQL or XQuery queries, optimization is reduced. The extent or size of the reduction depends on the machine size and the number of predicates.

When the query optimizer reduces the amount of query optimization, it continues to apply all the query rewrite rules that would normally be applied. However, it does use the Greedy join enumeration method and reduces the number of access plan combinations that are considered.

Query optimization class 5 is an excellent choice for a mixed environment consisting of both transactions and complex queries. This optimization class is designed to apply the most valuable query transformations and other query optimization techniques in an efficient manner.

7 - This class directs the optimizer to use a significant amount of optimization to generate an access plan. It is the same as query optimization class 5 except that it does not reduce the amount of query optimization for complex dynamic SQL or XQuery queries.

9 - This class directs the optimizer to use all available optimization techniques. These include:

- All available statistics
- All query rewrite rules
- All possibilities for join enumerations, including Cartesian products and unlimited composite inners
- All access methods

This class can greatly expand the number of possible access plans that are considered by the optimizer. You might use this class to find out whether more comprehensive optimization would generate a better access plan for very complex and very long-running queries that use large tables. Use Explain and performance measurements to verify that a better plan has actually been found.

Related concepts:

- "Choosing an optimization class" on page 153
- "Optimization strategies for intra-partition parallelism" on page 177
- "Optimization strategies for MDC tables" on page 18
- "Remote SQL generation and global optimization in federated databases" on page 217

Related tasks:

- "Setting the optimization class" on page 157

Setting the optimization class

When you specify an optimization level, consider whether a query uses static or dynamic SQL and XQuery statements, and whether the same dynamic query is repeatedly executed. For static SQL and XQuery statements, the query compilation time and resources are expended just once and the resulting plan can be used many times. In general, static SQL and XQuery statements should always use the default query optimization class. Because dynamic statements are bound and executed at run time, consider whether the overhead of additional optimization for dynamic statements improves overall performance. However, if the same dynamic SQL or XQuery statement is executed repeatedly, the selected access plan is cached. Such statements can use the same optimization levels as static SQL and XQuery statements.

If you think that a query that might benefit from additional optimization, but you are not sure, or you are concerned about compilation time and resource usage, you might perform some benchmark testing.

Procedure:

To specify a query optimization class, follow these steps:

1. Analyze the performance factors either informally or with formal tests as follows:
   - For **dynamic** query statements, tests should compare the average run time for the statement. Use the following formula to estimate an average run time:

     ```
     compile time + sum of execution times for all iterations
     -------------------------------------------------------
                   number of iterations
     ```

     In this formula, the `number of iterations` represents the number of times that you expect that the query statement might be executed each time it is compiled.

     Note: After the initial compilation, dynamic SQL and XQuery statements are recompiled when a change to the environment requires it. If the environment does not change after a query statement is cached, it does not need to be compiled again because subsequent PREPARE statements re-use the cached statement.
   - For **static** SQL and XQuery statements, compare the statement run times.

     Although you might also be interested in the compile time of static SQL and XQuery statements, the total compile and run time for the statement is difficult to assess in any meaningful context. Comparing the total time does not recognize the fact that a static query statement can be run many times for each time it is bound and that it is generally not bound during run time.
2. Specify the optimization class as follows:
   - **Dynamic** SQL and XQuery statements use the optimization class specified by the CURRENT QUERY OPTIMIZATION special register that you set with the SQL statement SET. For example, the following statement sets the optimization class to 1:

     ```
     SET CURRENT QUERY OPTIMIZATION = 1
     ```

To ensure that a dynamic SQL or XQuery statement always uses the same optimization class, you might include a SET statement in the application program.

If the CURRENT QUERY OPTIMIZATION register has not been set, dynamic statements are bound using the default query optimization class. The default value for both dynamic and static queries is determined by value of the database configuration parameter *dft_queryopt*. Class 5 is the default value of this parameter. The default values for the bind option and the special register are also read from the *dft_queryopt* database configuration parameter.

- **Static** SQL and XQuery statements use the optimization class specified on the PREP and BIND commands. The QUERYOPT column in the SYSCAT.PACKAGES catalog table records the optimization class used to bind the package. If the package is rebound either implicitly or using the REBIND PACKAGE command, this same optimization class is used for the static query statements. To change the optimization class for such static SQL and XQuery statements, use the BIND command. If you do not specify the optimization class, DB2 uses the default optimization as specified by *dft_queryopt* database configuration parameter.

Related concepts:

- "Choosing an optimization class" on page 153

Related reference:

- "Optimization classes" on page 154

Configuration parameters that affect query optimization

Several configuration parameters affect the access plan chosen by the SQL or XQuery compiler. Many of these are appropriate to a single-partition database environment and some are only appropriate to a partitioned database environment. In a partitioned database environment, the values used for each parameter should be the same on all database partitions.

Note: When you change a configuration parameter dynamically, the optimizer might not read the changed parameter values immediately because of older access plans in the package cache. To reset the package cache, execute the FLUSH PACKAGE CACHE command.

In a federated system, if the majority of your queries access nicknames, evaluate the types of queries that you send before you change your environment. For example, in a federated database the buffer pool does not cache pages from data sources, which are the DBMSs and data within the federated system. For this reason, increasing the size of the buffer does not guarantee that the optimizer will consider additional access-plan alternatives when it chooses an access plan for queries that contain nicknames. However, the optimizer might decide that local materialization of data source tables is the least-cost route or a necessary step for a sort operation. In that case, increasing the resources available to IBM DB2 V9.1 might improve performance.

The following configuration parameters or factors affect the access plan chosen by the SQL or XQuery compiler:

- The size of the buffer pools that you specified when you created or altered them.

  When the optimizer chooses the access plan, it considers the I/O cost of fetching pages from disk to the buffer pool and estimates the number of I/Os required to

satisfy a query. The estimate includes a prediction of buffer-pool usage, because additional physical I/Os are not required to read rows in a page that is already in the buffer pool.

The optimizer considers the value of the *npages* column in the BUFFERPOOLS system catalog tables and, in partitioned database environments, the BUFFERPOOLDBPARTITION system catalog tables.

The I/O costs of reading the tables can have an impact on:
- How two tables are joined
- Whether an unclustered index will be used to read the data

- Default Degree (dft_degree)

  The `dft_degree` configuration parameter specifies parallelism by providing a default value for the CURRENT DEGREE special register and the DEGREE bind option. A value of one (1) means no intra-partition parallelism. A value of minus one (-1) means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query.

- Default Query Optimization Class (dft_queryopt)

  Although you can specify a query optimization class when you compile SQL or XQuery queries, you might set a default optimization degree.

  Note: Intra-parallel processing does not occur unless you enable it by setting the *intra_parallel* database configuration parameter.

- Average Number of Active Applications (avg_appls)

  The optimizer uses the *avg_appls* parameter to help estimate how much of the buffer pool might be available at run-time for the access plan chosen. Higher values for this parameter can influence the optimizer to choose access plans that are more conservative in buffer pool usage. If you specify a value of 1, the optimizer considers that the entire buffer pool will be available to the application.

- Sort Heap Size (sortheap)

  If the rows to be sorted occupy more than the space available in the sort heap, several sort passes are performed, where each pass sorts a subset of the entire set of rows. Each sort pass is stored in a temporary table in the buffer pool, which might be written to disk. When all the sort passes are complete, these sorted subsets are merged into a single sorted set of rows. A sort is considered to be “piped” if it does not require a temporary table to store the final, sorted list of data. That is, the results of the sort can be read in a single, sequential access. Piped sorts result in better performance than non-piped sorts and will be used if possible.

  When choosing an access plan, the optimizer estimates the cost of the sort operations, including evaluating whether a sort can be piped, by:
  - Estimating the amount of data to be sorted
  - Looking at the *sortheap* parameter to determine if there is enough space for the sort to be piped.

- Maximum Storage for Lock List (locklist) and Maximum Percent of Lock List Before Escalation (maxlocks)

  When the isolation level is **repeatable read (RR)**, the optimizer considers the values of the *locklist* and *maxlocks* parameters to determine whether row level locks might be escalated to a table level lock. If the optimizer estimates that lock escalation will occur for a table access, then it chooses a table level lock for the access plan, instead of incurring the overhead of lock escalation during the query execution.

- CPU Speed (cpuspeed)

The optimizer uses the CPU speed to estimate the cost of performing certain operations. CPU cost estimates and various I/O cost estimates help select the best access plan for a query.

The CPU speed of a machine can have a significant influence on the access plan chosen. This configuration parameter is automatically set to an appropriate value when the database is installed or migrated. Do not adjust this parameter unless you are modelling a production environment on a test system or assessing the impact of a hardware change. Using this parameter to model a different hardware environment allows you to find out the access plans that might be chosen for that environment. To have DB2 recompute the value of this automatic configuration parameter, set it to -1.

- Statement Heap Size (stmtheap)

  Although the size of the statement heap does not influence the optimizer in choosing different access paths, it can affect the amount of optimization performed for complex SQL or XQuery statements.

  If the *stmtheap* parameter is not set large enough, you might receive a warning indicating that there is not enough memory available to process the statement. For example, SQLCODE +437 (SQLSTATE 01602) might indicate that the amount of optimization that has been used to compile a statement is less than the amount that you requested.

- Maximum Query Degree of Parallelism (max_querydegree)

  When the *max_querydegree* parameter has a value of ANY, the optimizer chooses the degree of parallelism to be used. If other than ANY is present, then the user-specified value determines the degree of parallelism for the application.

- Communications Bandwidth (comm_bandwidth)

  Communications bandwidth is used by the optimizer to determine access paths. The optimizer uses the value in this parameter to estimate the cost of performing certain operations between the database partition servers in a partitioned database environment.

- Application Heap Size (applheapsz)

  Large schemas require sufficient space in the application heap. It is recommended that you enlarge the application heap as follows:

  ```
  db2 update db cfg using applheapsz 1000
  ```

Related concepts:

- "Data-access methods" on page 164
- "Optimization strategies for intra-partition parallelism" on page 177
- "Optimization strategies for MDC tables" on page 18
- "The SQL and XQuery compiler process" on page 149

Related reference:

- "avg_appls - Average number of active applications " on page 424
- "comm_bandwidth - Communications bandwidth " on page 498
- "cpuspeed - CPU speed " on page 499
- "dft_degree - Default degree " on page 475
- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "max_querydegree - Maximum query degree of parallelism " on page 492
- "locklist - Maximum storage for lock list " on page 383
- "sortheap - Sort heap size " on page 400
- "stmtheap - Statement heap size " on page 402

Database database partition group impact on query optimization

In partitioned database environments, the optimizer recognizes collocation of tables and uses this collocation when it determines the best access plan for a query. If tables are frequently involved in join queries, they should be divided among database partitions in a partitioned database environment so that the rows from each table being joined are located on the same database partition. During the join operation, the collocation of the data from both joined tables prevents moving data from one database partition to another. Place both tables in the same database partition group to ensure that the data from the tables is collocated.

In a partitioned database environment, depending on the size of the table, spreading data over more database partitions reduces the estimated time (or cost) to execute a query. The number of tables, the size of the tables, the location of the data in those tables, and the type of query, such as whether a join is required, all affect the cost of the query.

Related concepts:

- "Join methods in partitioned database environments" on page 197
- "Join strategies in partitioned databases" on page 195
- "Adding database partitions in a partitioned database environment" in *Administration Guide: Implementation*

Table space impact on query optimization

Certain characteristics of your table spaces can affect the access plan chosen by the query compiler:

- Container characteristics

  Container characteristics can have a significant impact on the I/O cost associated during query execution. When it selects an access plan, the query optimizer considers these I/O costs, including any cost differences for accessing data from different table spaces. Two columns in the SYSCAT.TABLESPACES system catalog are used by the optimizer to help estimate the I/O costs of accessing data from a table space:

  – OVERHEAD, which provides an estimate in milliseconds of the time required by the container before any data is read into memory. This overhead activity includes the container's I/O controller overhead as well as the disk latency time, which includes the disk seek time.

    You may use the following formula to help you estimate the overhead cost:

    ```
    OVERHEAD = average seek time in milliseconds
               + (0.5 * rotational latency)
    ```

    where:
    - 0.5 represents an average overhead of one half rotation
    - Rotational latency is calculated in milliseconds for each full rotation, as follows:

      ```
      (1 / RPM) * 60 * 1000
      ```

      where you:
      - Divide by rotations per minute to get minutes per rotation
      - Multiply by 60 seconds per minute
      - Multiply by 1000 milliseconds per second.

As an example, let the rotations per minute for the disk be 7 200. Using the rotational-latency formula,this would produce:

```
(1 / 7200) * 60 * 1000 = 8.328 milliseconds
```

which can then be used in the calculation of the OVERHEAD estimate with an assumed average seek time of 11 milliseconds:

```
OVERHEAD = 11 + (0.5 * 8.328)
         = 15.164
```

giving an estimated OVERHEAD value of about 15 milliseconds.

– TRANSFERRATE, which provides an estimate in milliseconds of the time required to read one page of data into memory.

  If each table-space container is a single physical disk then you may use the following formula to help you estimate the transfer cost in milliseconds per page:

  ```
  TRANSFERRATE = (1 / spec_rate) * 1000 / 1 024 000 * page_size
  ```

  where:
  - `spec_rate` represents the disk specification for the transfer rate, in MB per second
  - Divide by `spec_rate` to get seconds per MB
  - Multiply by 1000 milliseconds per second
  - Divide by 1 024 000 bytes per MB
  - Multiply by the page size in bytes (for example, 4 096 bytes for a 4 KB page)

  As an example, suppose the specification rate for the disk is 3 MB per second. This would produce the following calculation

  ```
  TRANSFERRATE = (1 / 3) * 1000 / 1024000 * 4096
               = 1.333248
  ```

  giving an estimated TRANSFERRATE value of about 1.3 milliseconds per page.

  If the table space containers are not single physical disks but are arrays of disks (such as RAID), then you must take additional considerations into account when you attempt to determine the TRANSFERRATE to use. If the array is relatively small then you can multiply the `spec_rate` by the number of disks, assuming that the bottleneck is at the disk level.

  However, if the number of disks in the array making up the container is large, then the bottleneck may not be at the disk level, but at one of the other I/O subsystem components such as disk controllers, I/O busses, or the system bus. In this case, you cannot assume that the I/O throughput capability is the product of the `spec_rate` and the number of disks. Instead, you must measure the actual I/O rate in MBs during a sequential scan. For example, a sequential scan could be `select count(*) from big_table` and will be MBs in size. Divide the result by the number of containers that make up the table space in which `big_table` resides. Use the result as a substitute for `spec_rate` in the formula given above. For example, a measured sequential I/O rate of 100 MBs while scanning a table in a four container table space would imply 25 MBs per container, or a TRANSFERRATE of (1/25) * 1000 / 1024000 * 4096 = 0.16 milliseconds per page.

Each of the containers assigned to a table space may reside on different physical disks. For best results, all physical disks used for a given table space should have the same OVERHEAD and TRANSFERRATE characteristics. If these

characteristics are not the same, you should use the average when setting the values for OVERHEAD and TRANSFERRATE.

You can obtain media-specific values for these columns from the hardware specifications or through experimentation. These values may be specified on the CREATE TABLESPACE and ALTER TABLESPACE statements.

Experimentation becomes especially important in the environment mentioned above where you have a disk array as a container. You should create a simple query that moves data and use it in conjunction with a platform-specific measuring utility. You can then re-run the query with different container configurations within your table space. You can use the CREATE and ALTER TABLESPACE statements to change how data is transferred in your environment.

The I/O cost information provided through these two values could influence the optimizer in a number of ways, including whether or not to use an index to access the data, and which table to select for the inner and outer tables in a join.

- Prefetching

  When considering the I/O cost of accessing data from a table space, the optimizer also considers the potential impact that prefetching data and index pages from disk can have on the query performance. Prefetching data and index pages can reduce the overhead and wait time associated with reading the data into the buffer pool.

  The optimizer uses the information from the PREFETCHSIZE and EXTENTSIZE columns in SYSCAT.TABLESPACES to estimate the amount of prefetching that will occur for a table space.

  – EXTENTSIZE can only be set when creating a table space (for example using the CREATE TABLESPACE statement). The default extent size is 32 pages (of 4 KB each) and is usually sufficient.
  – PREFETCHSIZE can be set when you create a table space and or use the ALTER TABLESPACE statement. The default prefetch size is determined by the value of the DFT_PREFETCH_SZ database configuration parameter. Review the recommendations for sizing this parameter and make changes as needed to improve the data movement.

The following shows an example of the syntax to change the characteristics of the RESOURCE table space:

```
ALTER TABLESPACE RESOURCE
  PREFETCHSIZE 64
  OVERHEAD     19.3
  TRANSFERRATE 0.9
```

After making any changes to your table spaces, consider rebinding your applications and executing the RUNSTATS utility to collect the latest statistics about the indexes to ensure that the best access plans are used.

Related concepts:

- "Database managed space" in *Administration Guide: Planning*
- "Catalog statistics tables" on page 286
- "The SQL and XQuery compiler process" on page 149

Optimizing Access Plans

Data-access methods

When it compiles an SQL or XQuery statement, the query optimizer estimates the execution cost of different ways of satisfying the query. Based on its estimates, the optimizer selects an optimal access plan. An *access plan* specifies the order of operations required to resolve an SQL or XQuery statement. When an application program is bound, a *package* is created. This package contains access plans for all of the static SQL and XQuery statements in that application program. Access plans for dynamic SQL and XQuery statements are created at the time that the application is executed.

There are two ways to access data in a table:

- Scanning the entire table sequentially
- Locating specific table rows by first accessing an index on the table

To produce the results that the query requests, rows are selected depending on the terms of the predicate, which are usually stated in a WHERE clause. The selected rows in accessed tables are joined to produce the result set, and the result set might be further processed by grouping or sorting the output.

Related concepts:

- "Data access through index scans" on page 164
- "Index access and cluster ratios" on page 169
- "The SQL and XQuery compiler process" on page 149
- "Types of index access" on page 167

Data access through index scans

An *index scan* occurs when the database manager accesses an index for any of the following reasons:

- To narrow the set of qualifying rows (by scanning the rows in a certain range of the index) before accessing the base table. The index *scan range* (the start and stop points of the scan) is determined by the values in the query against which index columns are being compared.
- To order the output.
- To retrieve the requested column data directly. If all of the requested data is in the index, the indexed table does not need to be accessed. This is known as an *index-only access*.

If indexes are created with the ALLOW REVERSE SCANS option, scans may also be performed in the direction opposite to that with which they were defined.

Note: The optimizer chooses a table scan if no appropriate index has been created or if an index scan would be more costly. An index scan might be more costly when the table is small the index-clustering ratio is low, or the query requires most of the table rows. To find out whether the access plan uses a table scan or an index scan, use the Explain facility.

Index Scans to Delimit a Range

To determine whether an index can be used for a particular query, the optimizer evaluates each column of the index starting with the first column to see if it can be used to satisfy equality and other predicates in the WHERE clause. A *predicate* is an element of a search condition in a WHERE clause that expresses or implies a comparison operation. Predicates that can be used to delimit the range of an index scan in the following cases:

- Tests for equality against a constant, a host variable, an expression that evaluates to a constant, or a keyword
- Tests for "IS NULL" or "IS NOT NULL"
- Tests for equality against a basic subquery, which is a subquery that does not contain ANY, ALL, or SOME, and the subquery does not have a correlated column reference to its immediate parent query block (that is, the SELECT for which this subquery is a subselect).
- Tests for strict and inclusive inequality.

The following examples illustrate when an index might be used to limit a range:

- Consider an index with the following definition:

  ```
  INDEX IX1:  NAME    ASC,
              DEPT    ASC,
              MGR     DESC,
              SALARY  DESC,
              YEARS   ASC
  ```

  In this case, the following predicates might be used to limit the range of the scan of index IX1:

  ```
  WHERE  NAME = :hv1
    AND  DEPT = :hv2
  ```

  or

  ```
  WHERE  MGR  = :hv1
    AND  NAME = :hv2
    AND  DEPT = :hv3
  ```

  Note that in the second WHERE clause, the predicates do not have to be specified in the same order as the key columns appear in the index. Although the examples use host variables, other variables such as parameter markers, expressions, or constants would have the same effect.

- Consider a single index created using the ALLOW REVERSE SCANS parameter. Such indexes support scans in the direction defined when the index was created as well as in the opposite or reverse direction. The statement might look something like this:

  ```
  CREATE INDEX iname ON tname (cname DESC) ALLOW REVERSE SCANS
  ```

  In this case, the index (`iname`) is formed based on DESCending values in `cname`. By allowing reverse scans, although the index on the column is defined for scans in descending order, a scan can be done in ascending order. The actual use of the index in both directions is not controlled by you but by the optimizer when creating and considering access plans.

In the following WHERE clause, only the predicates for NAME and DEPT would be used in delimiting the range of the index scan, but not the predicates for SALARY or YEARS:

```
WHERE  NAME   = :hv1
  AND  DEPT   = :hv2
  AND  SALARY = :hv4
  AND  YEARS  = :hv5
```

This is because there is a key column (MGR) separating these columns from the first two index key columns, so the ordering would be off. However, once the range is determined by the NAME = :hv1 and DEPT = :hv2 predicates, the remaining predicates can be evaluated against the remaining index key columns.

Index Scans to Test Inequality

Certain inequality predicates can delimit the range of an index scan. There are two types of inequality predicates:

- Strict inequality predicates

  The strict inequality operators used for range delimiting predicates are greater than (>) and less than (<).

  Only one column with strict inequality predicates is considered for delimiting a range for an index scan. In the following example, the predicates on the NAME and DEPT columns can be used to delimit the range, but the predicate on the MGR column cannot be used.

  ```
  WHERE  NAME   = :hv1
    AND  DEPT   > :hv2
    AND  DEPT   < :hv3
    AND  MGR    < :hv4
  ```

- Inclusive inequality predicates

  The following are inclusive inequality operators that can be used for range delimiting predicates:

  – >= and <=
  – BETWEEN
  – LIKE

  For delimiting a range for an index scan, multiple columns with inclusive inequality predicates will be considered. In the following example, all of the predicates can be used to delimit the range of the index scan:

  ```
  WHERE  NAME   = :hv1
    AND  DEPT  >= :hv2
    AND  DEPT  <= :hv3
    AND  MGR   <= :hv4
  ```

  To further illustrate this example, suppose that `:hv2 = 404`, `:hv3 = 406`, and `:hv4 = 12345`. The database manager will scan the index for all of departments 404 and 405, but it will stop scanning department 406 when it reaches the first manager that has an employee number (MGR column) greater than 12345.

Index Scans to Order Data

If the query requires output in sorted order, an index might be used to order the data if the ordering columns appear consecutively in the index, starting from the first index key column. Ordering or sorting can result from operations such as ORDER BY, DISTINCT, GROUP BY, "= ANY" subquery, "> ALL" subquery, "< ALL" subquery, INTERSECT or EXCEPT, UNION. An exception to this is when the index key columns are compared for equality against "constant values", which is any expression that evaluates to a constant. In this case the ordering column can be other than the first index key columns.

Consider the following query:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY MGR
```

For this query, the index might be used to order the rows because NAME and DEPT will always be the same values and will thus be ordered. That is, the preceding WHERE and ORDER BY clauses are equivalent to:

```
WHERE NAME = 'JONES'
  AND DEPT = 'D93'
ORDER BY NAME, DEPT, MGR
```

A unique index can also be used to truncate a sort-order requirement. Consider the following index definition and ORDER BY clause:

```
UNIQUE INDEX IX0:  PROJNO  ASC
SELECT PROJNO, PROJNAME, DEPTNO
  FROM PROJECT
ORDER BY PROJNO, PROJNAME
```

Additional ordering on the PROJNAME column is not required because the IX0 index ensures that PROJNO is unique. This uniqueness ensures that there is only one PROJNAME value for each PROJNO value.

Related concepts:

- "Data-access methods" on page 164
- "Index access and cluster ratios" on page 169
- "Index structure" on page 26
- "Types of index access" on page 167

Types of index access

In some cases, the optimizer might find that all data that a query requires from a table can be retrieved from an index on the table. In other cases, the optimizer might use more than one index to access tables. In the case of range-clustered tables, data can be accessed via a ″virtual″ index, which computes the location of data records.

Index-Only Access

In some cases, all of the required data can be retrieved from the index without accessing the table. This is known as an *index-only* access.

To illustrate an index-only access, consider the following index definition:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

The following query can be satisfied by accessing only the index, and without reading the base table:

```
SELECT NAME, DEPT, MGR, SALARY
  FROM EMPLOYEE
 WHERE NAME = 'SMITH'
```

Often, however, required columns that do not appear in the index. To obtain the data for these columns, the table rows must be read. To allow the optimizer to choose an index-only access, create a unique index with include columns. For example, consider the following index definition:

```
CREATE UNIQUE INDEX IX1 ON EMPLOYEE
   (NAME ASC)
    INCLUDE (DEPT, MGR, SALARY, YEARS)
```

This index enforces uniqueness of the NAME column and also stores and maintains data for DEPT, MGR, SALARY, and YEARS columns, which allows the following query to be satisfied by accessing only the index:

```
SELECT NAME, DEPT, MGR, SALARY
   FROM EMPLOYEE
   WHERE NAME='SMITH'
```

When you consider adding INCLUDE columns to an index, however, consider whether the additional storage space and maintenance costs are justified. If queries that can be satisfied by reading only such an index are rarely executed, the costs might not be justified.

Multiple Index Access

The optimizer can choose to scan multiple indexes on the same table to satisfy the predicates of a WHERE clause. For example, consider the following two index definitions:

```
INDEX IX2:  DEPT    ASC
INDEX IX3:  JOB     ASC,
            YEARS   ASC
```

The following predicates can be satisfied by using the two indexes:

```
WHERE DEPT = :hv1
  OR (JOB   = :hv2
  AND YEARS >= :hv3)
```

Scanning index IX2 produces a list of row IDs (RIDs) that satisfy the `DEPT = :hv1` predicate. Scanning index IX3 produces a list of RIDs satisfying the `JOB = :hv2 AND YEARS >= :hv3` predicate. These two lists of RIDs are combined and duplicates removed before the table is accessed. This is known as *index ORing*.

Index ORing may also be used for predicates specified in the IN clause, as in the following example:

```
WHERE DEPT IN (:hv1, :hv2, :hv3)
```

Although the purpose of index ORing is to eliminate duplicate RIDs, the objective of *index ANDing* is to find common RIDs. Index ANDing might occur with applications that create multiple indexes on corresponding columns in the same table and a query using multiple AND predicates is run against that table. Multiple index scans against each indexed column in such a query produce values which are hashed to create bitmaps. The second bitmap is used to probe the first bitmap to generate the qualifying rows that are fetched to create the final returned data set.

For example, given the following two index definitions:

```
INDEX IX4: SALARY   ASC
INDEX IX5: COMM     ASC
```

the following predicates could be resolved using these two indexes:

```
WHERE SALARY BETWEEN 20000 AND 30000
   AND COMM BETWEEN 1000 AND 3000
```

In this example, scanning index IX4 produces a bitmap satisfying the SALARY BETWEEN 20000 AND 30000 predicate. Scanning IX5 and probing the bitmap for IX4 results in the list of qualifying RIDs that satisfy both predicates. This is known as "dynamic bitmap ANDing". It occurs only if the table has sufficient cardinality and the columns have sufficient values in the qualifying range, or sufficient duplication if equality predicates are used.

To realize the performance benefits of dynamic bitmaps when scanning multiple indexes, it may be necessary to change the value of the sort heap size (*sortheap*) database configuration parameter, and the sort heap threshold (*sheapthres*) database manager configuration parameter.

Additional sort heap space is required when dynamic bitmaps are used in access plans. When *sheapthres* is set to be relatively close to *sortheap* (that is, less than a factor of two or three times per concurrent query), dynamic bitmaps with multiple index access must work with much less memory than the optimizer anticipated. The solution is to increase the value of *sheapthres* relative to *sortheap*.

Note: The optimizer does not combine index ANDing and index ORing in accessing a single table.

Index Access in Range clustered tables

Unlike standard tables, a range clustered table does not require a physical index that maps a key value to a row like a traditional B-tree index. Instead, it leverages the sequential nature of the column domain and uses a functional mapping to generate the location of a given row in a table. In the simplest example of this mapping, the first key value in the range is the first row in the table, and the second value in the range is the second row in the table, and so on.

The optimizer uses the range-clustered property of the table to generate access plans based on a perfectly clustered index whose only cost is computing the range clustering function. The clustering of rows within the table is guaranteed because range clustered tables retain their original key value ordering.

Related concepts:

- "Data-access methods" on page 164
- "Data access through index scans" on page 164
- "Index access and cluster ratios" on page 169
- "Using relational indexes to improve performance" on page 25

Index access and cluster ratios

When it chooses an access plan, the optimizer estimates the number of I/Os required to fetch required pages from disk to the buffer pool. This estimate includes a prediction of buffer-pool usage, since additional I/Os are not required to read rows in a page that is already in the buffer pool.

For index scans, information from the system catalog tables (SYSCAT.INDEXES) helps the optimizer estimate I/O cost of reading data pages into the buffer pool. It uses information from the following columns in the SYSCAT.INDEXES table:

- CLUSTERRATIO information indicates the degree to which the table data is clustered in relation to this index. The higher the number, the better rows are ordered in index key sequence. If table rows are in close to index-key sequence,

rows can be read from a data page while the page is in the buffer. If the value of this column is -1, the optimizer uses PAGE_FETCH_PAIRS and CLUSTERFACTOR information if it is available.
- PAGE_FETCH_PAIRS contains pairs of numbers that model the number of I/Os required to read the data pages into buffer pools of various sizes together with CLUSTERFACTOR information. Data is collected for these columns only if you execute RUNSTATS on the index with the DETAILED clause.

If index clustering statistics are not available, the optimizer uses default values, which assume poor clustering of the data to the index.

The degree to which the data is clustered with respect to the index can have a significant impact on performance and you should try to keep one of the indexes on the table close to 100 percent clustered.

In general, only one index can be one hundred percent clustered, except in those cases where the keys are a superset of the keys of the clustering index or where there is de facto correlation between the key columns of the two indexes.

When you reorganize an table, you can specify an index that will be used to cluster the rows and attempt to preserve this characteristic during insert processing. Because updates and inserts may make the table less well clustered in relation to the index, you might need to periodically reorganize the table. To reduce the frequency of reorganization on a table that has frequent changes due to INSERTs, UPDATEs, and DELETES, use the PCTFREE parameter when you alter a table. This allows for additional inserts to be clustered with the existing data.

Related concepts:
- "Relational index performance tips" on page 30
- "Types of index access" on page 167

Predicate typology and access plans

A user application requests a set of rows from the database with a query statement that specifies qualifiers for the specific rows to be returned as the result set. These qualifiers usually appear in the WHERE clause of the query. Such qualifiers are called *predicates*. Predicates can be grouped into four categories that are determined by how and when the predicate is used in the evaluation process. The categories are listed below, ordered in terms of performance from best to worst:

1. Range delimiting predicates
2. Index SARGable predicates
3. Data SARGable predicates
4. Residual predicates.

Note: *SARGable* refers to a term that can be used as a search *arg*ument.

The following table summarizes the predicate categories. Subsequent sections describe each category in more detail.

Table 49. Summary of Predicate Type Characteristics

Characteristic	Predicate Type			
	Range Delimiting	**Index SARGable**	**Data SARGable**	**Residual**
Reduce index I/O	Yes	No	No	No
Reduce data page I/O	Yes	Yes	No	No
Reduce number of rows passed internally	Yes	Yes	Yes	No
Reduce number of qualifying rows	Yes	Yes	Yes	Yes

Range-Delimiting and Index-SARGable Predicates

Range delimiting predicates limit the scope of an index scan. They provide start and stop key values for the index search. Index SARGable predicates cannot limit the scope of a search, but can be evaluated from the index because the columns involved in the predicate are part of the index key. For example, consider the following index:

```
INDEX IX1:  NAME    ASC,
            DEPT    ASC,
            MGR     DESC,
            SALARY  DESC,
            YEARS   ASC
```

Consider also a query that contains the following WHERE clause:

```
WHERE  NAME  = :hv1
 AND   DEPT  = :hv2
 AND   YEARS > :hv5
```

The first two predicates (NAME = :hv1, DEPT = :hv2) are range-delimiting predicates, while YEARS > :hv5 is an index SARGable predicate.

The optimizer uses the index data when it evaluates these predicates instead of reading the base table. These *index SARGable* predicates reduce the set of rows that need to be read from the table, but they do not affect the number of index pages that are accessed.

Predicates on XML data occuring in XMLEXISTS and XMLTABLE expressions are also supported by XSCAN data operator scans. Some of these predicates are also supported by index range scans.

Data SARGable Predicates

Predicates that cannot be evaluated by the index manager, but can be evaluated by data management services are called *data SARGable* predicates. These predicates usually require accessing individual rows from a table. If required, Data Management Services retrieve the columns needed to evaluate the predicate, as well as any others to satisfy the columns in the SELECT list that could not be obtained from the index.

For example, consider a single index defined on the PROJECT table:

```
INDEX IX0:  PROJNO ASC
```

For the following query, then, the `DEPTNO = 'D11'` predicate is considered to be data SARGable.

```
SELECT PROJNO, PROJNAME, RESPEMP
  FROM PROJECT
 WHERE DEPTNO = 'D11'
 ORDER BY PROJNO
```

Residual Predicates

Residual predicates require more I/O costs than accessing a table. They might have the following characteristics:

- Use correlated subqueries
- Use quantified subqueries, which contain ANY, ALL, SOME, or IN clauses
- Read LONG VARCHAR or LOB data, which is stored in a file that is separate from the table

Such predicates are evaluated by Relational Data Services.

Sometimes predicates that are applied only to the index must be reapplied when the data page is accessed. For example, access plans that use index ORing or index ANDing always reapply the predicates as residual predicates when the data page is accessed.

Related concepts:

- "The SQL and XQuery compiler process" on page 149

Effects of sorting and grouping

When the optimizer chooses an access plan, it considers the performance impact of sorting data. Sorting occurs when no index satisfies the requested ordering of fetched rows. Sorting might also occur when the optimizer determines that a sort is less expensive than an index scan. The optimizer sort data in one of the following ways:

- Piping the results of the sort when the query is executed.
- Internal handling of the sort within the database manager.

Piped versus non-piped sorts

If the final sorted list of data can be read in a single sequential pass, the results can be *piped*. Piping is quicker than non-piped ways of communicating the results of the sort. The optimizer chooses to pipe the results of a sort whenever possible.

Whether or not a sort is piped, the sort time depends on a number of factors, including the number of rows to be sorted, the key size and the row width. If the rows to be sorted occupy more than the space available in the sort heap, several sort passes are performed, in which each pass sorts a subset of the entire set of rows. Each sort pass is stored in a temporary table in the buffer pool. If there is not enough space in the buffer pool, pages from this temporary table might be written to disk. When all the sort passes are complete, these sorted subsets must be merged into a single sorted set of rows. If the sort is piped, the rows are handed directly to Relational Data Services as they are merged.

Group and sort pushdown operators

In some cases, the optimizer can choose to push down a sort or aggregation operation to Data Management Services from the Relational Data Services component. Pushing down these operations improves performance by allowing the Data Management Services component to pass data directly to a sort or aggregation routine. Without this pushdown, Data Management Services first passes this data to Relational Data Services, which then interfaces with the sort or aggregation routines. For example, the following query benefits from this optimization:

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_DEPT_SALARY
  FROM EMPLOYEE
  GROUP BY WORKDEPT
```

Group operations in sorts

When sorting produces the order required for a GROUP BY operation, the optimizer can perform some or all of the GROUP BY aggregations while doing the sort. This is advantageous if the number of rows in each group is large. It is even more advantageous if doing some of the grouping during the sort reduces or eliminates the need for the sort to spill to disk.

An aggregation in sort requires as many as the following three stages of aggregation to ensure that proper results are returned.

1. The first stage of aggregation, partial aggregation, calculates the aggregate values until the sort heap is filled. In partial aggregation unaggregated data is taken in and partial aggregates are produced. If the sort heap is filled, the rest of the data spills to disk, including all of the partial aggregations that have been calculated in the current sort heap. After the sort heap is reset, new aggregations are started.
2. The second stage of aggregation, intermediate aggregation, takes all of the spilled sort runs, and aggregates further on the grouping keys. The aggregation cannot be completed because the grouping key columns are a subset of the distribution key columns. Intermediate aggregation uses existing partial aggregates to produce new partial aggregates. This stage does not always occur. It is used for both intra-partition and inter-partition parallelism. In intra-partition parallelism, the grouping is finished when a global grouping key is available. In inter-partition parallelism, this occurs when the grouping key is a subset of the distribution key dividing groups across database partitions, and thus requires redistribution to complete the aggregation. A similar case exists in intra-partition parallelism when each agent finishes merging its spilled sort runs before reducing to a single agent to complete the aggregation.
3. The last stage of aggregation, final aggregation, uses all of the partial aggregates and produces final aggregates. This step always takes place in a GROUP BY operator. Sort cannot perform complete aggregation because they cannot guarantee that the sort will not split. Complete aggregation takes in unaggregated data and produces final aggregates. If the distribution does not prohibit its use, this method of aggregation is usually used to group data that is already in the correct order.

Related concepts:

- "Tuning sort performance" on page 111

Related reference:

- "sheapthres - Sort heap threshold " on page 399

- "sortheap - Sort heap size " on page 400

Using index and column group statistics to compute grouping keycard

When a query requires data to be grouped in a certain way, the optimizer needs to compute the number of distinct groupings, or *grouping keycard*. A grouping requirement can result from operations such as GROUP BY or DISTINCT.

Consider the following query:

```
SELECT DEPTNO, YEARS, AVG(SALARY)
  FROM EMPLOYEE
 GROUP BY DEPTNO, MGR, YEAR_HIRED
```

Without any index or column group statistics, the number of groupings (also in this case the number of rows returned) estimated by the optimizer will be the product of the number of distinct values of DEPTNO, MGR, and YEAR_HIRED. This estimate assumes that the grouping key columns are independent. However, this assumption could be erroneous if each manager manages exactly one department. Also, it is unlikely that each department has employees hired every year. Thus, the product of distinct values of DEPTNO, MGR, and YEAR_HIRED could be an overestimate of the actual number of distinct groups.

Now, consider an index with the following definition:

```
INDEX IX1:  DEPTNO, MGR, YEAR_HIRED
```

In this case, the FULLKEYCARD of IX1 provides the optimizer with the exact number of distinct groupings for the query above.

Consider another index definition:

```
INDEX IX2:  DEPTNO, MGR, YEAR_HIRED, COMM
```

IX2 could also help to compute the grouping keycard, since its FIRST3KEYCARD indicates how many distinct groups of (DEPTNO,MGR,YEAR_HIRED).

Besides index statistics (FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, and FULLKEYCARD), column group statistics could also be exploited by the optimizer to compute grouping keycard in a similar fashion. Column group statistics collected on DEPTNO, MGR, and YEAR_HIRED will provide the same benefit as IX1 and IX2 above:

```
RUNSTATS ON TABLE EMPLOYEE ON COLUMNS ((DEPTNO, MGR, YEAR_HIRED))
```

Note that if the grouping keys consist of five or more columns, then collecting column group statistics might be preferable. This is because RUNSTATS only collects statistics on the first four columns and the full index key columns of any given index.

Related concepts:

- "Column correlation for multiple predicates" on page 193
- "View statistics relevant to optimization" on page 182

Related tasks:

- "Collecting distribution statistics for specific columns" on page 281

Improving performance by binding with REOPT

SQL or XQuery queries may perform poorly during execution if the values used for the input variables such as parameter markers, host variables, and special registers, are outside the predictive range of default filter factor estimates. Default filter factors, used for scenarios where the actual data value is not known, are estimates of how many rows will actually be returned at runtime when the actual data value is used.

The REOPT bind option specifies whether or not to have DB2 optimize an access path using values for host variables, parameter markers, and special registers. REOPT values are specified by the following arguments to the BIND, PREP and REBIND commands:

REOPT NONE
The access path for a given SQL or XQuery statement containing host variables, parameter markers, or special registers will not be optimized using real values for these variables; The default estimates for the these variables are used instead. This plan is cached and will be used subsequently. This is the default behavior.

REOPT ONCE
The access path for a given SQL or XQuery statement will be optimized using the real values of the host variables, parameter markers, or special registers when the query is first executed. This plan is cached and used subsequently.

REOPT ALWAYS
The access path for a given SQL or XQuery statement will always be compiled and reoptimized using the values of the host variables, parameter markers, or special registers known at each execution time.

Related concepts:

- "Performance improvements when using REOPT option of the BIND command" in *Developing Embedded SQL Applications*

Replicated materialized query tables in partitioned database environments

Replicated materialized query tables improve performance of frequently executed joins in a partitioned database environment by allowing the database to manage precomputed values of the table data.

Consider an example of a query and a replicated materialized table. The following assumptions are made:

- The SALES table is in the multipartition table space REGIONTABLESPACE, and is split on the REGION column.
- The EMPLOYEE and DEPARTMENT tables are in a single-partition database partition group.

Create a replicated materialized query table based on the information in the EMPLOYEE table.

```
CREATE TABLE R_EMPLOYEE
  AS (
      SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
      FROM   EMPLOYEE
```

```
    )
DATA INITIALLY DEFERRED REFRESH IMMEDIATE
IN REGIONTABLESPACE
REPLICATED;
```

To update the content of the replicated materialized query table, run the following statement:

```
REFRESH TABLE R_EMPLOYEE;
```

Note: After using the REFRESH statement, you should run RUNSTATS on the replicated table as you would any other table.

The following example calculates sales by employee, the total for the department, and the grand total:

```
SELECT d.mgrno, e.empno, SUM(s.sales)
FROM   department AS d, employee AS e, sales AS s
WHERE  s.sales_person = e.lastname
   AND e.workdept = d.deptno
GROUP BY ROLLUP(d.mgrno, e.empno)
ORDER BY d.mgrno, e.empno;
```

Instead of using the EMPLOYEE table, which is on only one database partition, the database manager uses the R_EMPLOYEE table, which is replicated on each of the database partitions where the SALES tables is stored. The performance enhancement occurs because the employee information does not have to be moved across the network to each database partition to calculate the join.

Replicated materialized query tables in collocated joins

Replicated materialized query tables can also assist in the collocation of joins. For example, if a star schema contains a large fact table spread across twenty nodes, the joins between the fact table and the dimension tables are most efficient if these tables are collocated. If all of the tables are in the same database partition group, at most one dimension table is partitioned correctly for a collocated join. The other dimension tables cannot be used in a collocated join because the join columns on the fact table do not correspond to the distribution key of the fact table.

Consider a table called FACT (C1, C2, C3, ...) split on C1; and a table called DIM1 (C1, dim1a, dim1b, ...) split on C1; and a table called DIM2 (C2, dim2a, dim2b, ...) split on C2; and so on.

In this case, you see that the join between FACT and DIM1 is perfect because the predicate DIM1.C1 = FACT.C1 is collocated. Both of these tables are split on column C1.

However, the join between DIM2 with the predicate WHERE DIM2.C2 = FACT.C2 cannot be collocated because FACT is split on column C1 and not on column C2. In this case, you might replicate DIM2 in the database partition group of the fact table so that the join occurs locally on each database partition.

Note: The replicated materialized query tables discussion here is related to intra-database replication. Inter-database replication is concerned with subscriptions, control tables, and data located in different databases and on different operating systems.

When you create a replicated materialized query table, the source table can be a single-node table or a multi-node table in a database partition group. In most

cases, the replicated table is small and can be placed in a single-node database partition group. You can limit the data to be replicated by specifying only a subset of the columns from the table or by specifying the number of rows through the predicates used, or by using both methods. The data capture option is not required for replicated materialized query tables to function.

A replicated materialized query table can also be created in a multi-node database partition group so that copies of the source table are created on all of the database partitions. Joins between a large fact table and the dimension tables are more likely to occur locally in this environment that if you broadcast the source table to all database partitions.

Indexes on replicated tables are not created automatically. You can create indexes that are different from those on the source table. However, to prevent constraint violations that are not present on the source tables, you cannot create unique indexes or put constraints on the replicated tables. Constraints are disallowed even if the same constraint occurs on the source table.

Replicated tables can be referenced directly in a query, but you cannot use the NODENUMBER() predicate with a replicated table to see the table data on a particular partition.

Use the EXPLAIN facility to see if a replicated materialized query table was used by the access plan for a query. Whether the access plan chosen by the optimizer uses the replicated materialized query table depends on the information that needs to be joined. The optimizer might not use the replicated materialized query table if the optimizer determines that it would be cheaper to broadcast the original source table to the other database partitions in the database partition group.

Related concepts:

• "Joins" on page 187

Optimization strategies for intra-partition parallelism

The optimizer can choose an access plan to execute a query in parallel within a single database partition if a degree of parallelism is specified when the SQL statement is compiled.

At execution time, multiple database agents called subagents are created to execute the query. The number of subagents is less than or equal to the degree of parallelism specified when the SQL statement was compiled.

To parallelize an access plan, the optimizer divides it into a portion that is run by each subagent and a portion that is run by the coordinating agent. The subagents pass data through table queues to the coordinating agent or to other subagents. In a partitioned database environment, subagents can send or receive data through table queues from subagents in other database partitions.

Intra-partition parallel scan strategies

Relational scans and index scans can be performed in parallel on the same table or index. For parallel relational scans, the table is divided into ranges of pages or rows. A range of pages or rows is assigned to a subagent. A subagent scans its assigned range and is assigned another range when it has completed its work on the current range.

For parallel index scans, the index is divided into ranges of records based on index key values and the number of index entries for a key value. The parallel index scan proceeds like the parallel table scan with subagents being assigned a range of records. A subagent is assigned a new range when it has complete its work on the current range.

The optimizer determines the scan unit (either a page or a row) and the scan granularity.

Parallel scans provide an even distribution of work among the subagents. The goal of a parallel scan is to balance the load among the subagents and keep them equally busy. If the number of busy subagents equals the number of available processors and the disks are not overworked with I/O requests, then the machine resources are being used effectively.

Other access plan strategies might cause data imbalance as the query executes. The optimizer chooses parallel strategies that maintain data balance among subagents.

Intra-partition parallel sort strategies

The optimizer can choose one of the following parallel sort strategies:

- **Round-robin sort**

  This is also known as a *redistribution sort*. This method uses shared memory efficiently redistribute the data as evenly as possible to all subagents. It uses a round-robin algorithm to provide the even distribution. It first creates an individual sort for each subagent. During the insert phase, subagents insert into each of the individual sorts in a round-robin fashion to achieve a more even distribution of data.
- **Partitioned sort**

  This is similar to the round-robin sort in that a sort is created for each subagent. The subagents apply a hash function to the sort columns to determine into which sort a row should be inserted. For example, if the inner and outer tables of a merge join are a partitioned sort, a subagent can use merge join to join the corresponding table portions and execute in parallel.
- **Replicated sort**

  This sort is used if each subagent requires all of the sort output. One sort is created and subagents are synchronized as rows are inserted into the sort. When the sort is completed, each subagent reads the entire sort. If the number of rows is small, this sort may be used to rebalance the data stream.
- **Shared sort**

  This sort is the same as a replicated sort, except the subagents open a parallel scan on the sorted result to distribute the data among the subagents in a way similar to the round-robin sort.

Intra-partition parallel temporary tables

Subagents can cooperate to produce a temporary table by inserting rows into the same table. This is called a `shared temporary table`. The subagents can open private scans or parallel scans on the shared temporary table depending on whether the data stream is to be replicated or split.

Intra-partition parallel aggregation strategies

Aggregation operations can be performed in parallel by subagents. An aggregation operation requires the data to be ordered on the grouping columns. If a subagent can be guaranteed to receive all the rows for a set of grouping column values, it can perform a complete aggregation. This can happen if the stream is already split on the grouping columns because of a previous partitioned sort.

Otherwise, the subagent can perform a partial aggregation and use another strategy to complete the aggregation. Some of these strategies are:

- Send the partially aggregated data to the coordinator agent through a merging table queue. The coordinator completes the aggregation.
- Insert the partially aggregated data into a partitioned sort. The sort is split on the grouping columns and guarantees that all rows for a set of grouping columns are contained in one sort partition.
- If the stream needs to be replicated to balance processing, the partially aggregated data can be inserted into a replicated sort. Each subagent completes the aggregation using the replicated sort, and receives an identical copy of the aggregation result.

Intra-partition parallel join strategies

Join operations can be performed in parallel by subagents. Parallel join strategies are determined by the characteristics of the data stream.

A join can be parallelized by partitioning or by replicating the data stream on the inner and outer tables of the join, or both. For example, a nested loop join can be parallelized if its outer stream is partitioned for a parallel scan and the inner stream is re-evaluated independently by each subagent. A merged join can be parallelized if its inner and outer streams are value-partitioned for partitioned sorts.

Related concepts:

- "Optimization strategies for MDC tables" on page 18

Statistical views

Statistical views

Statistical views allow the optimizer to compute more accurate cardinality estimates. Cardinality estimation is the process whereby the optimizer uses statistics to determine the size of partial query results after predicates are applied or aggregation is performed. The accuracy of cardinality estimates depends on the predicates and the available statistics. Statistics are available to represent the distribution of data values within a column, which can improve cardinality estimates when the data values are unevenly distributed. Statistics are also available to represent the number of distinct values in a set of columns, which can improve cardinality estimates when columns are statistically correlated. However, quite often, these statistics may not be able to represent more complex relationships such as the filtering effect of predicates or aggregation involving correlated and skewed attributes (for example, *make* = 'Honda' AND *model* = 'Accord'), comparisons with expression results (for example, *price* > MSRP + Dealer_markup), relationships spanning multiple tables (for example *product.name* = 'Alloy wheels' and *product.key = sales.product_key*), or anything other than predicates or aggregation involving independent attributes and simple comparison operations.

Statistical views are views with associated statistics that can be used to improve cardinality estimates for queries in which the view definition overlaps with the query definition. This is a powerful feature in that it provides the optimizer with accurate statistics for determining cardinality estimates for queries with complex sets of (possibly correlated) predicates involving one or more tables.

A statistical view need not be directly referenced in a query it optimizes; in most cases, a view's statistics can be exploited if its definition overlaps with the query definition. To exploit this new feature, a view must be enabled for optimization using the ALTER VIEW statement and system catalog tables must be populated for statistics on the view.

Related concepts:

- "Materialized query tables" on page 11
- "Scenario: Improving cardinality estimates using statistical views" on page 182

Related tasks:

- "Using statistical views" on page 180

Related reference:

- "RUNSTATS command" in *Command Reference*

Using statistical views

A view must be enabled for optimization before its statistics can be used to optimize a query. A view that is enabled for optimization is a *statistical view*. A view that is not a statistical view is said to be disabled for optimization. The term *regular view* is used to refer to a view that is disabled for optimization. A view is disabled for optimization when it is first created.

Prerequisites:

- To enable a view for optimization you must have ALTER privilege on both the view and the table on which the view is defined.
- To invoke RUNSTATS for a view you must have one of the following:
  - SYSADM
  - SYSCTRL
  - SYSMAINT
  - DBADM
  - CONTROL privilege on the view

  In addition, you need to have appropriate privileges to access rows from the view. Specifically, for each table, view or nickname referenced in the view definition, you must have one of the following privileges:
  - SYSADM
  - DBADM
  - CONTROL
  - SELECT

Restrictions:

A view cannot be enabled for optimization if any of the following are true:

- The view directly or indirectly references an MQT. (An MQT or statistical view can reference a statistical view.)
- It is an inoperative view.
- It is a typed view.
- There is another view alteration in the same ALTER VIEW statement.

If the definition of a view that is altered to enable optimization meets any of the conditions below, the ALTER VIEW ENABLE OPTIMIZATION will succeed with a warning, but the optimizer will not exploit its statistics:

- It contains aggregation or distinct operations.
- It contains union, except, or intersect operations.
- It contains scalar aggregate (OLAP) functions.

Procedure:

1. Enable the view for optimization. A view can be enabled for optimization using the new ENABLE OPTIMIZATION clause on the ALTER VIEW statement. A view that has been enabled for optimization can subsequently be disabled for optimization using the DISABLE OPTIMIZATION clause on the ALTER VIEW statement. For example, to enable the view `myview` for optimization, enter the following:

   ```
   ALTER VIEW myview ENABLE QUERY OPTIMIZATION
   ```

   A view that is enabled for optimization has a 'Y' in character position 13 of the PROPERTY column of its corresponding SYSTABLES entry. A view that is disabled for optimization has a blank in character position 13 of the PROPERTY column of its corresponding SYSTABLES entry.
2. Execute RUNSTATS. For example, to collect statistics on the view `myview`, enter the following:

   ```
   RUNSTATS ON TABLE db2dba.myview
   ```

   To use statistics sampling to collect view statistics, including distribution statistics, on 10 percent of the rows using row-level sampling, enter:

   ```
   RUNSTATS ON TABLE db2dba.myview WITH DISTRIBUTION TABLESAMPLE BERNOULLI (10)
   ```

   Note: Prior to DB2 Version 9.1, executing RUNSTATS on a statistical view only primed the catalog statistics tables for manual updates and did not collect any statistics. In DB2 Version 9.1, executing RUNSTATS on a statistical view will collect statistics. This means that RUNSTATS may take much longer to execute than previously, depending on how much data is returned by the view.
3. Updating view statistics can result in changes to the plans for queries that overlap the view definition. If these queries are part of static SQL packages, these packages must be rebound to take advantages of the plans resulting from the new statistics.

Related concepts:

- "Statistical views" on page 179
- "View statistics relevant to optimization" on page 182

Related reference:

- "RUNSTATS command" in *Command Reference*

View statistics relevant to optimization

Only statistics that characterize the data distribution of the query defining a statistical view, such as CARD and COLCARD, are considered for optimization by the optimizer. The following statistics associated with view records can be collected and can be exploited by the optimizer.

Table statistics (SYSCAT.TABLES, SYSSTAT.TABLES)

- CARD - The number of rows in the view result.

Column statistics (SYSCAT.COLUMNS, SYSSTAT.COLUMNS)

- COLCARD - The number of distinct values of a column in the view result
- AVGCOLLEN - Average length of column in the view result
- HIGH2KEY - Second highest value of a column in the view result
- LOW2KEY - Second lowest value of a column in the view result
- NUMNULLS - Number of NULLs in a view result column
- SUB_COUNT - Average number of sub-elements in a view result column
- SUB_DELIM_LENGTH - Average length of each delimiter separating each sub-element

Column Distribution Statistics (SYSCAT.COLDIST, SYSSTAT.COLDIST)

- DISTCOUNT - If TYPE is Q, the number of distinct values that are less than or equal to COLVALUE statistics
- SEQNO - Frequency ranking of a sequence number to help uniquely identify the row in the table
- COLVALUE - Data value for which frequency or quantile statistic is collected
- VALCOUNT - Frequency with which the data value occurs in view column, or for quantiles, the number of values less than or equal to the data value (COLVALUE)

Statistics that do not describe data distribution, such as NPAGES, and FPAGES, can be collected, but will be ignored by the optimizer.

Related concepts:

- "Catalog statistics" on page 277
- "Statistical views" on page 179
- "Sub-element statistics" on page 295

Related reference:

- "RUNSTATS command" in *Command Reference*

Scenario: Improving cardinality estimates using statistical views

In a data warehouse, fact table information often changes quite dynamically, while dimension table data is static. This means that dimension attribute data might be positively or negatively correlated with fact table attribute data. Traditional base table statistics currently available to the optimizer do not allow it to discern relationships across tables. Column and table distribution statistics on statistical views (and MQTs) can be used to give the optimizer the necessary information to correct these types of cardinality estimation errors.

Consider the following query which computes annual sales revenue for golf clubs sold during July of each year:

```
SELECT  sum(f.sales_price), d2.year
FROM  product d1, period d2, daily_sales f
WHERE  d1.prodkey = f.prodkey
    AND  d2.perkey = f.perkey
    AND  d1.item_desc = 'golf club'
    AND  d2.month = 'JUL'
GROUP BY  d2.year
```

A star join query execution plan can be an excellent choice for this query provided the optimizer can determine whether the semi-join involving PRODUCT and DAILY_SALES, or the semi-join involving PERIOD and DAILY_SALES, is the most selective. In order to generate an efficient star join plan, the optimizer must be able to choose the most selective semi-join for the outer leg of the index anding operation.

Data warehouses often contain records for products that are no longer on store shelves. This can cause the distribution of PRODUCT columns after the join to appear dramatically different than their distribution before the join. Since the optimizer, for lack of better information, will determine the selectivity of local predicates based solely on base table statistics, the optimizer might become overly optimistic regarding the selectivity of the predicate *item_desc = 'golf club'*

For example, if golf clubs represent 1% of the products manufactured historically, but accounts now for 20% of recent sales, the optimizer would likely overestimate the selectivity of *item_desc = 'golf club'*, as there are no statistics describing the distribution of *item_desc* after the join. And if sales in all twelve months are equally likely, the selectivity of the predicate *month = 'JUL'* would be around 8%, and thus the error in estimating the selectivity of the predicate *item_desc = 'golf club'* would mistakenly cause the optimizer to perform the seemingly more selective semi-join between PRODUCT and DAILY_SALES as the outer leg of the star join plan's index anding operation.

The following example provides a step-by-step illustration of how to set up statistical views to solve this type of problem.

The following is a database from a typical data warehouse, where STORE, CUSTOMER, PRODUCT, PROMOTION, and PERIOD are the dimension tables, and DAILY_SALES the fact table.

Table 50. STORE (63 rows)

Column	storekey	store_number	city	state	district	...
Attribute	integer	char(2)	char(20)	char(5)	char(14)	...
	not null					
	primary key					

Table 51. CUSTOMER (1,000,000 rows)

Column	custkey	name	address	age	gender	...
Attribute	integer	char(30)	char(40)	smallint	char(1)	...
	not null					
	primary key					

Table 52. PRODUCT (19,450 rows)

Column	prodkey	category	item_desc	price	cost	...
Attribute	integer	integer	char(30)	decimal(11)	decimal(11)	...
	not null					
	primary key					

Table 53. PROMOTION (35 rows)

Column	promokey	promotype	promodesc	promovalue	...
Attribute	integer	integer	char(30)	decimal(5)	...
	not null				
	primary key				

Table 54. PERIOD (2922 rows)

Column	perkey	calendar_date	month	period	year	...
Attribute	integer	date	char(3)	smallint	smallint	...
	not null					
	primary key					

Table 55. DAILY_SALES (754 069 426 rows)

Column	storekey	custkey	prodkey	promokey	perkey	sales_price	...
Attribute	integer	integer	integer	integer	integer	decimal(11)	...

Suppose the company managers wish to find out whether or not consumers will buy a product again if they are offered a discount on the same product on a return visit. In addition, suppose the study is only done for store '01', which has 18 locations nationwide. Table 56 shows the different categories of promotion available, annotated by the percentage of promotions.

Table 56. PROMOTION (35 rows)

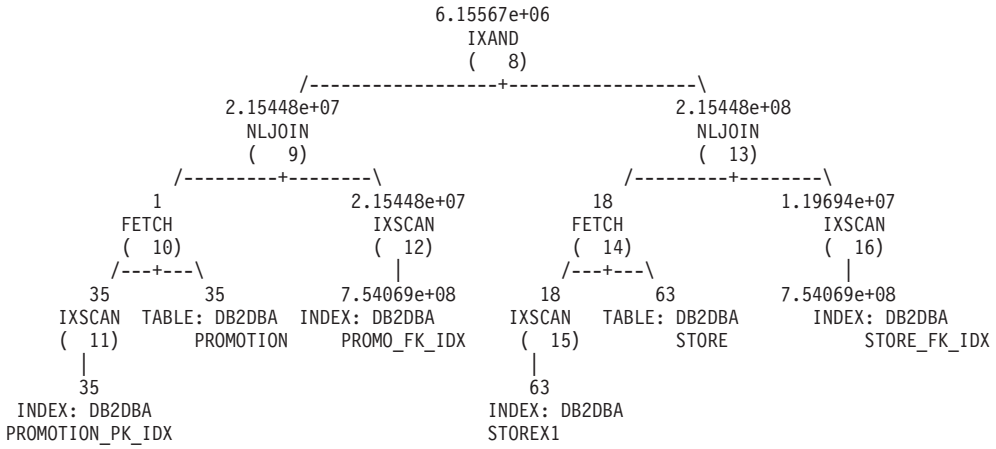
promotype	promodesc	COUNT(promotype)	percentage of promotions
1	Return customers	1	2.86%
2	Coupon	15	42.86%
3	Advertisement	5	14.29%
4	Manager's special	3	8.57%
5	Overstocked items	4	11.43%
6	End aisle display	7	20.00%

The table indicates that only 2.86% of all 35 kinds of promotions offered comes from discounts for return customers (1÷35 ≈ 0.0286) .

The following query is then executed and a count of 12,889,514 is returned:

```
SELECT  count(*)
FROM  store d1, promotion d2, daily_sales f
WHERE  d1.storekey = f.storekey
    AND  d2.promokey = f.promokey
    AND  d1.store_number = '01'
    AND  d2.promotype = 1
```

This query is executed according to the following plan generated by the optimizer. In each node of this diagram, the top number represents the cardinality estimate, the second row is the operator type, and the number in parentheses is the operator id.



At the nested loop join number 9, the optimizer estimates that around 2.86% of the product sold resulted from customers coming back to buy the same products at a discounted price (2.15448e+07 ÷7.54069e+08 ≈ 0.0286). Note that this is same percentage before and after joining the PROMOTION table with the DAILY_SALES table. Table 57 summarizes the cardinality estimates and their percentage (the filtering effect) before and after the join.

Table 57. Cardinality estimates before and after joining with DAILY_SALES.

	Before Join		**After Join**	
Predicates	**count**	**percentage of rows qualified**	**count**	**percentage of rows qualified**
store_number = '01'	18	28.57%	2.15448e+08	28.57%
promotype=1	1	2.86%	2.15448e+07	2.86%

Because the probability of *promotype = 1* is less than that of *store_number = '01'*, the optimizer chooses the semi-join between PROMOTION and DAILY_SALES as the outer leg of the star join plan's index anding operation. This leads to an estimated count of approximately 6,155,670 products sold using promotion type 1 — an incorrect cardinality estimate off by a factor of 2.09 (12,889,514 ÷ 6,155,670 ≈ 2.09).

What causes the optimizer to only be able to estimate half of the actual number of records qualifying the two predicates? Store '01' represents about 28.57% of all the stores. What if other stores had more sales than store '01' (less than 28.57%)? Or what if store '01' actually sold most of the product (more than 28.57%)? Likewise, the 2.86% of products sold using promotion type 1 shown in Table 57 can be misleading. The actual percentage in DAILY_SALES could very well be a different figure than the projected one.

We can use statistical views to help the optimizer correct its estimates and reduce overestimation or underestimation. First, we need to create two statistical views representing each semi-join in the query above. The first statistical view provides the distribution of promotion types for all daily sales. The second statistical view represents the distribution of promotion types for all daily sales. Note that each statistical view can provide the distribution information for any particular store

number or promotion type. In this example, we use a 10% sample rate to retrieve the records in DAILY_SALES for the respective views and save them in global temporary tables. We then query those tables to collect the necessary statistics to update the two statistical views.

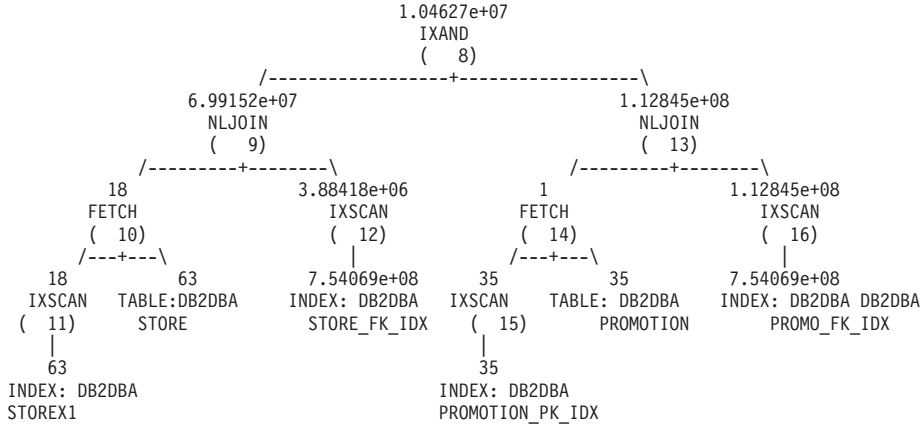
1. Create a view representing the join of STORE with DAILY_SALES.

```
CREATE  view sv_store_dailysales as
   (SELECT  s.*
    FROM  store s, daily_sales ds
    WHERE  s.storekey = ds.storekey)
```

2. Create a view representing the join of PROMOTION with DAILY_SALES.

```
CREATE  view sv_promotion_dailysales as
   (SELECT  p.*
    FROM  promotion.p, daily_sales ds
    WHERE  p.promokey = ds.promokey)
```

3. Make the views statistical views by enabling them for query optimization:
   - `ALTER VIEW  sv_store_dailysales  ENABLE QUERY OPTIMIZATION`
   - `ALTER VIEW  sv_promotion_dailysales  ENABLE QUERY OPTIMIZATION`
4. Execute RUNSTATS to collect statistics on the views:
   - `RUNSTATS  on table db2dba.sv_store_dailysales  WITH DISTRIBUTION`
   - `RUNSTATS  on table db2dba.sv_promotion_dailysales WITH DISTRIBUTION`
5. Run the query again so that it can be re-optimized. Upon reoptimization, the optimizer will match SV_STORE_DAILYSALES and SV_PROMOTION_DAILYSALES with the query and will use the view statistics to adjust the cardinality estimate of the semi-joins between the fact and dimension tables, causing it to reverse the original order of the semi-joins chosen without these statistics. The new plan is as follows:



Note that this time, the semi-join between STORE and DAILY_SALES is performed on the outer leg of the index anding plan. This is because the two statistical views essentially tell the optimizer that the predicate store_number = '01' will filter more rows than promotype = 1. Table 58 on page 187 summarizes the cardinality estimates and their percentage (filtering effect) before and after the join for each semi-join.

Note that this time, the semi-join between STORE and DAILY_SALES is performed on the outer leg of the index anding plan. This is because the two statistical views essentially tell the optimizer that the predicate store_number = '01' will filter more rows than promotype = 1. Here, the optimizer estimates that there are approximately 10,462,700 products sold. This estimate is off by a factor of 1.23 (12,889,514 ÷ 10,462,700 ≈ 1.23), which is a significant improvement over the

estimate in Table 57 on page 185. Table 58 summarizes the cardinality estimates and the filtering effect for each predicate before and after the join.

Table 58. Cardinality estimates before and after joining with DAILY_SALES.

	Before Join		**After Join** *no statistical views*		**After Join** *with statistical views*	
Predicates	**count**	**percentage of rows qualified**	**count**	**percentage of rows qualified**	**count**	**percentage of rows qualified**
store_number = '01'	18	28.57%	2.15448e+08	28.57%	6.99152e+07	9.27%
promotype=1	1	2.86%	2.15448e+07	2.86%	1.12845e+08	14.96%

This time, the optimizer estimates that there are approximately 10,462,700 products sold. This estimate is off by a factor of 1.23, which is a significant improvement over the estimate without statistical views.

Related concepts:

- "Collecting statistics on a sample of the table data" on page 285
- "Statistical views" on page 179
- "Strategies for selecting optimal joins" on page 191

Related tasks:

- "Collecting catalog statistics" on page 279
- "Using statistical views" on page 180

Related reference:

- "RUNSTATS command" in *Command Reference*

Joins

Joins

A *join* is the process of combining information from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the joining criterion.

For example, consider the following two tables:

Table1

PROJ	**PROJ_ID**
A	1
B	2
C	3
D	4

Table2

PROJ_ID	**NAME**
1	Sam
3	Joe
4	Mary
1	Sue
2	Mike

To join Table1 and Table2 where the ID columns have the same values, use the following SQL statement:

```
SELECT PROJ, x.PROJ_ID, NAME
  FROM TABLE1 x, TABLE2 y
  WHERE x.PROJ_ID = y.PROJ_ID
```

This query yields the following set of result rows:

PROJ	PROJ_ID	NAME
A	1	Sam
A	1	Sue
B	2	Mike
C	3	Joe
D	4	Mary

Depending on the existence of a join predicate, as well as various costs involved as determined by table and index statistics, the optimizer chooses one of the following join methods:

- Nested-loop join
- Merge join
- Hash join

When two tables are joined, one table is selected as the outer table and the other as the inner. The outer table is accessed first and is scanned only once. Whether the inner table is scanned multiple times depends on the type of join and the indexes that are present. Even if a query joins more than two tables, the optimizer joins only two tables at a time. If necessary, temporary tables are created to hold intermediate results.

You can provide explicit join operators, such as INNER or LEFT OUTER JOIN to determine how tables are used in the join. Before you alter a query in this way, however, you should allow the optimizer to determine how to join the tables. Then analyze query performance to decide whether to add join operators.

Related concepts:

- "Join information" on page 642
- "Join methods" on page 188
- "Join methods in partitioned database environments" on page 197
- "Join strategies in partitioned databases" on page 195

Join methods

The optimizer can choose one of three basic join strategies when queries require tables to be joined.

- Nested-loop join
- Merge join
- Hash join

These methods are described in the following sections.

Nested-Loop Join

A nested-loop join is performed in one of the following two ways:

- Scanning the inner table for each accessed row of the outer table

For example, consider that column A in tables T1 and T2 have the following values:

Outer table T1: column A	Inner table T2: column A
2	3
3	2
3	2
	3
	1

To perform a nested-loop join, the database manager performs the following steps:

1. Read the first row from T1. The value for A is "2"
2. Scan T2 until a match ("2") is found, and then join the two rows
3. Scan T2 until the next match ("2") is found, and then join the two rows
4. Scan T2 to the end of the table
5. Go back to T1 and read the next row ("3")
6. Scan T2, starting at the first row, until a match ("3") is found, and then join the two rows
7. Scan T2 until the next match ("3") is found, and then join the two rows
8. Scan T2 to the end of the table
9. Go back to T1 and read the next row ("3")
10. Scan T2 as before, joining all rows which match ("3").

- Performing an index lookup on the inner table for each accessed row of the outer table

  This method can be used for the specified predicates if there is a predicate of the following form:

  ```
  expr(outer_table.column)  relop  inner_table.column
  ```

  where `relop` is a relative operator (for example =, >, >=, <, or <=) and `expr` is a valid expression on the outer table. Consider the following examples:

  ```
  OUTER.C1 + OUTER.C2 <= INNER.C1
  OUTER.C4 < INNER.C3
  ```

  This method might significantly reduce the number of rows accessed in the inner table for each access of the outer table, although it depends on a number of factors, including the selectivity of the join predicate.

When it evaluates a nested loop join, the optimizer also decides whether to sort the outer table before performing the join. If it orders the outer table, based on the join columns, the number of read operations to access pages from disk for the inner table might be reduced, because they are more likely to be be in the buffer pool already. If the join uses a highly clustered index to access the inner table and if the outer table has been sorted, the number of index pages accessed might be minimized.

In addition, if the optimizer expects that the join will make a later sort more expensive, it might also choose to perform the sort before the join. A later sort might be required to support a GROUP BY, DISTINCT, ORDER BY or merge join.

Merge Join

Merge join, sometimes known as *merge scan join* or *sort merge join*, requires a predicate of the form table1.column = table2.column. This is called an *equality join*

predicate. Merge join requires ordered input on the joining columns, either through index access or by sorting. A merge join cannot be used if the join column is a LONG field column or a large object (LOB) column.

In a merge join, the joined tables are scanned at the same time. The outer table of the merge join is scanned only once. The inner table is also scanned once unless repeated values occur in the outer table. If there are repeated values occur, a group of rows in the inner table might be scanned again. For example, if column A in tables T1 and T2 has the following values:

Outer table T1: column A	Inner table T2: column A
2	1
3	2
3	2
	3
	3

To perform a merge join, the database manager performs the following steps:

1. Read the first row from T1. The value for A is "2".
2. Scan T2 until a match is found, and then join the two rows.
3. Keep scanning T2 while the columns match, joining rows.
4. When the "3" in T2 is read, go back to T1 and read the next row.
5. The next value in T1 is "3", which matches T2, so join the rows.
6. Keep scanning T2 while the columns match, joining rows.
7. The end of T2 is reached.
8. Go back to T1 to get the next row — note that the next value in T1 is the same as the previous value from T1, so T2 is scanned again starting at the first "3" in T2. The database manager remembers this position.

Hash Join

A hash join requires one or more predicates of the form table1.columnX = table2.columnY, for which the column types are the same. For columns of type CHAR, the length must be the same. For columns of type DECIMAL, the precision and scale must be the same. The column type cannot be a LONG field column, or a large object (LOB) column.

First, the designated INNER table is scanned and the rows copied into memory buffers drawn from the sort heap specified by the *sortheap* database configuration parameter. The memory buffers are divided into sections based on a hash value that is computed on the columns of the join predicates. If the size of the INNER table exceeds the available sort heap space, buffers from selected sections are written to temporary tables.

When the inner table has been processed, the second, or OUTER, table is scanned and its rows are matched to rows from the INNER table by first comparing the hash value computed for the columns of the join predicates. If the hash value for the OUTER row column matches the hash value of the INNER row column, the actual join predicate column values are compared.

OUTER table rows that correspond to portions of the table not written to a temporary table are matched immediately with INNER table rows in memory. If the corresponding portion of the INNER table was written to a temporary table,

the OUTER row is also written to a temporary table. Finally, matching pairs of table portions from temporary tables are read, and the hash values of their rows are matched, and the join predicates are checked.

For the full performance benefits of hash join, you might need to change the value of the *sortheap* database configuration parameter and the *sheapthres* database manager configuration parameter.

For the full performance benefits of hash joins, you might need to change the value of the *sortheap* database configuration parameter and the *sheapthres* database manager configuration parameter.

Hash-join performance is best if you can avoid hash loops and overflow to disk. To tune hash-join performance, estimate the maximum amount of memory available for *sheapthres*, then tune the **sortheap** parameter. Increase its setting until you avoid as many hash loops and disk overflows as possible, but do not reach the limit specified by the *sheapthres* parameter.

Increasing the *sortheap* value should also improve performance of queries that have multiple sorts.

Related concepts:

- "Join information" on page 642
- "Join methods in partitioned database environments" on page 197
- "Join strategies in partitioned databases" on page 195
- "Joins" on page 187

Related reference:

- "sortheap - Sort heap size " on page 400
- "sheapthres - Sort heap threshold " on page 399

Strategies for selecting optimal joins

The optimizer uses various methods to select an optimal join strategy for a query. Among these methods are the following search strategies, which are determined by the optimization class of the query:

- Greedy join enumeration
  - Efficient with respect to space and time
  - Single direction enumeration; that is, once a join method is selected for two tables, it is not changed during further optimization
  - Might miss the best access plan when joining many tables. If your query joins only two or three tables, the access plan chosen by the greedy join enumeration is the same as the access plan chosen by dynamic programming join enumeration. This is particularly true if the query has many join predicates on the same column, either explicitly specified, or implicitly generated through predicate transitive closure.
- Dynamic programming join enumeration
  - Space and time requirements increase exponentially as the number of joined tables increases
  - Efficient and exhaustive search for best access plan
  - Similar to the strategy used by DB2 for OS/390 or z/OS.

The join-enumeration algorithm is an important determinant of the number of plan combinations that the optimizer explores.

Star-Schema Joins

The tables referenced in a query are almost always related by join predicates. If two tables are joined without a join predicate, the Cartesian product of the two tables is formed. In a Cartesian product, every qualifying row of the first table is joined with every qualifying row of the second, creating a result table consisting of the cross product of the size of the two tables that is usually very large. Since such a plan is unlikely to perform well, the optimizer avoids even determining the cost of such an access plan.

The only exceptions occur when the optimization class is set to 9 or in the special case of star schemas. A *star schema* contains a central table called the fact table and the other tables are called dimension tables. The dimension tables all have only a single join that attaches them to the fact table, regardless of the query. Each dimension table contains additional values that expand information about a particular column in the fact table. A typical query consists of multiple local predicates that reference values in the dimension tables and contains join predicates connecting the dimension tables to the fact table. For these queries it might be beneficial to compute the Cartesian product of multiple small dimension tables before accessing the large fact table. This technique is beneficial when multiple join predicates match a multi-column index.

DB2 can recognize queries against databases designed with star schemas that have at least two dimension tables and can increase the search space to include possible plans that compute the Cartesian product of dimension tables. If the plan that computes the Cartesian products has the lowest estimated cost, it is selected by the optimizer.

The star schema join strategy discussed above assumes that primary key indexes are used in the join. Another scenario involves foreign key indexes. If the foreign key columns in the fact table are single-column indexes and there is a relatively high selectivity across all dimension tables, the following star join technique can be used:

1. Process each dimension table by:
   - Performing a semi-join between the dimension table and the foreign key index on the fact table
   - Hashing the row ID (RID) values to dynamically create a bitmap.
2. Use AND predicates against the previous bitmap for each bitmap.
3. Determine the surviving RIDs after processing the last bitmap.
4. Optionally sort these RIDs.
5. Fetch a base table row.
6. Rejoin the fact table with each of its dimension tables, accessing the columns in dimension tables that are needed for the SELECT clause.
7. Reapply the residual predicates.

This technique does not require multi-column indexes. Explicit referential-integrity constraints between the fact table and the dimension tables are not required, although the relationship between the fact table and the dimension tables should actually be related in this way.

The dynamic bitmaps created and used by star join techniques require sort heap memory, the size of which is specified by the Sort Heap Size (*sortheap*) database configuration parameter.

Composite Tables

When the result of joining a pair of tables is a new table known as a *composite* table, this table usually becomes the outer table of another join with another inner table. This is known as a "composite outer" join. In some cases, particularly when using the greedy-join enumeration technique, it is useful to make the result of joining two tables the inner table of a later join. When the inner table of a join consists of the result of joining two or more tables, this plan is a "composite inner" join. For example, consider the following query:

```
SELECT COUNT(*)
FROM T1, T2, T3, T4
WHERE T1.A = T2.A AND
      T3.A = T4.A AND
      T2.Z = T3.Z
```

It might be beneficial to join table T1 and T2 (T1xT2), then join T3 to T4 (T3xT4) and finally select the first join result as the outer table and the second join result as the inner table. In the final plan ((T1xT2) x (T3xT4)), the join result (T3xT4) is known as a composite inner. Depending on the query optimization class, the optimizer places different constraints on the maximum number of tables that may be the inner table of a join. Composite inner joins are allowed with optimization classes 5, 7, and 9.

Related concepts:

- "Join methods" on page 188
- "Join methods in partitioned database environments" on page 197
- "Join strategies in partitioned databases" on page 195
- "Joins" on page 187

Column correlation for multiple predicates

Your applications might contain queries that are constructed with joins such that more than one join predicate joins two tables. This is not unusual when queries need to determine relationships between similar, related columns in different tables.

For example, consider a manufacturer who makes products from raw material of various colors, elasticities and qualities. The finished product has the same color and elasticity as the raw material from which it is made. The manufacturer issues the query:

```
     SELECT PRODUCT.NAME, RAWMATERIAL.QUALITY
FROM PRODUCT, RAWMATERIAL
        WHERE PRODUCT.COLOR       =  RAWMATERIAL.COLOR
          AND PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

This query returns the names and raw material quality of all products. There are two join predicates:

```
   PRODUCT.COLOR       =  RAWMATERIAL.COLOR
   PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

When the optimizer chooses a plan for executing this query, it calculates how selective each of the two predicates is. It assumes that they are independent, which means that all variations of elasticity occur for each color, and, conversely, that for each level of elasticity there is raw material of every color. It then estimates the overall selectivity of the pair of predicates by using catalog statistic information for each table based on the number of levels of elasticity and the number of different colors. Based on this estimate, it may choose, for example, a nested loop join in preference to a merge join, or vice versa.

However, it may be that these two predicates are not independent. For example, it may be that the highly elastic materials are available in only a few colors, and the very inelastic materials are only available in a few other colors that are different from the elastic ones. Then the combined selectivity of the predicates eliminates fewer rows so the query will return more rows. Consider the extreme case, in which there is just one level of elasticity for each color and vice versa. Now either one of the predicates logically could be omitted entirely since it is implied by the other. The optimizer might no longer choose the best plan. For example, it might choose a nested loop join plan when the merge join would be faster.

The DB2 optimizer attempts to detect and compensate for correlation of join predicates if you define an index on those columns or if you collect and maintain group column statistics on the appropriate columns.

For example, in elasticity example above, you might define one or both of the following indexes:

```
IX1 PRODUCT.COLOR, PRODUCT.ELASTICITY
```

or

```
IX2 RAWMATERIAL.COLOR, RAWMATERIAL.ELASTICITY
```

or both.

For the optimizer to detect correlation, the non-include columns of this index must be only the correlated columns. The index may also contain include columns to allow index-only scans. If there are more than two correlated columns in join predicates, make sure that you define the index to cover all of them.

One condition that must be met for the optimizer to consider an index key cardinality to detect correlation is that the number of distinct values in each column must be higher for each column from the same table. For example, imagine you have defined IX1 and IX2 as above. If the number of distinct values in PRODUCT.COLOR is less than the number of distinct values in RAWMATERIAL.COLOR and the number of distinct values in PRODUCT.ELASTICITY is less than that of RAWMATERIAL.ELASTICITY, then the optimizer will use IX2 to detect the correlation. Comparing column cardinalities makes it likely, but not certain, that the distinct values in the PRODUCT columns are included in the distinct values of the RAWMATERIAL columns. To make it even more likely that one domain assumes another, the optimizer might also compare the HIGH2KEY AND LOW2KEY statistics for these index columns.

After creating appropriate indexes, ensure that statistics on tables are accurate and up to date.

The optimizer uses the information in the FIRSTnKEYCARD and FULLKEYCARD columns of the unique index statistics table to detect cases of correlation, and

dynamically adjust combined selectivities of the correlated predicates, thus obtaining a more accurate estimate of the join size and cost.

As an alternative, column group statistics can be collected on a set of columns. In the elasticity example above, you might gather statistics on the columns PRODUCT.COLOR, PRODUCT.ELASTICITY and/or RAWMATERIAL.COLOR, RAWMATERIAL.ELASTCITY.

Column group statistics are collected using the ″ON COLUMNS″ option of RUNSTATS. For example, to collect the column group statistics on PRODUCT.COLOR and PRODUCT.ELASTICITY, issue the following RUNSTATS command:

```
RUNSTATS ON TABLE product ON COLUMNS ((color, elasticity))
```

Correlation of simple equal predicates

In addition to JOIN predicate correlation, the optimizer also manages correlation with simple equal predicates of the type COL =. For example, consider a table of different types of cars, each having a MAKE (that is, a manufacturer), MODEL, YEAR, COLOR, and STYLE, such as sedan, station wagon, sports-utility vehicle. Because almost every manufacturer makes the same standard colors available on each of their models and styles, year after year, predicates on COLOR are likely to be independent of those on MAKE, MODEL, STYLE, or YEAR. However, the predicates MAKE and MODEL certainly are not independent since only a single car maker would make a model with a particular name. Identical model names used by two or more car makers is very unlikely and certainly not wanted by the car makers.

If an index exists on the two columns MAKE and MODEL or column group statistics are gathered, the optimizer uses the statistical information about the index or columns to determine the combined number of distinct values and adjust the selectivity or cardinality estimation for correlation between the two columns. If the predicates are local equality predicates, the optimizer does not need a unique index to make the adjustment.

Related concepts:

- “Query rewriting methods and examples” on page 203
- “The SQL and XQuery compiler process” on page 149
- “Using index and column group statistics to compute grouping keycard” on page 174

Related tasks:

- “Collecting distribution statistics for specific columns” on page 281

Related reference:

- “RUNSTATS command” in *Command Reference*

Join strategies in partitioned databases

In some ways, join strategies are different in a partitioned database environment than in a non-partitioned database environment. Additional techniques can be applied to standard join methods to improve performance.

One consideration for those tables involved in frequent joins in a partitioned database environment is that of table collocation. Table collocation provides the means in a partitioned database environment to locate data from one table with the data from another table at the same database partition based on the same distribution key. Once collocated, data to be joined can participate in a query without having to be moved to another database partition as part of the query activity. Only the answer set for the join is moved to the coordinator node.

Table Queues

The descriptions of join techniques in a partitioned database environment use the following terminology:

- table queue

  A mechanism for transferring rows between database partitions, or between processors in a single partition database.
- directed table queue

  A table queue in which rows are hashed to one of the receiving database partitions.
- broadcast table queue

  A table queue in which rows are sent to all of the receiving database partitions, but are not hashed.

A table queue is used in the following circumstances:

- To pass table data from one database partition to another when using inter-partition parallelism
- To pass table data within a database partition when using intra-partition parallelism
- To pass table data within a database partition when using a single partition database.

Each table queue is passes the data in a single direction. The compiler decides where table queues are required, and includes them in the plan. When the plan is executed, the connections between the database partitions initiate the table queues. The table queues close as processes end.

There are several types of table queues:

- *Asynchronous table queues.* These table queues are known as asynchronous because they read rows in advance of any FETCH being issued by the application. When the FETCH is issued, the row is retrieved from the table queue.

  Asynchronous table queues are used when you specify the FOR FETCH ONLY clause on the SELECT statement. If you are only fetching rows, the asynchronous table queue is faster.
- *Synchronous table queues.* These table queues are known as synchronous because they read one row for each FETCH that is issued by the application. At each database partition, the cursor is positioned on the next row to be read from that database partition.

  Synchronous table queues are used when you do not specify the FOR FETCH ONLY clause on the SELECT statement. In a partitioned database environment, if you are updating rows, the database manager will use the synchronous table queues.
- *Merging table queues.*

  These table queues preserve order.

- *Non-merging table queues.*

  These table queues are also known as "regular" table queues. They do not preserve order.
- *Listener table queues.*

  These table queues are use with correlated subqueries. Correlation values are passed down to the subquery and the results are passed back up to the parent query block using this type of table queue.

Related concepts:

- "Join methods" on page 188
- "Join methods in partitioned database environments" on page 197
- "Joins" on page 187

Join methods in partitioned database environments

The following figures illustrate join methods in a partitioned database environment.

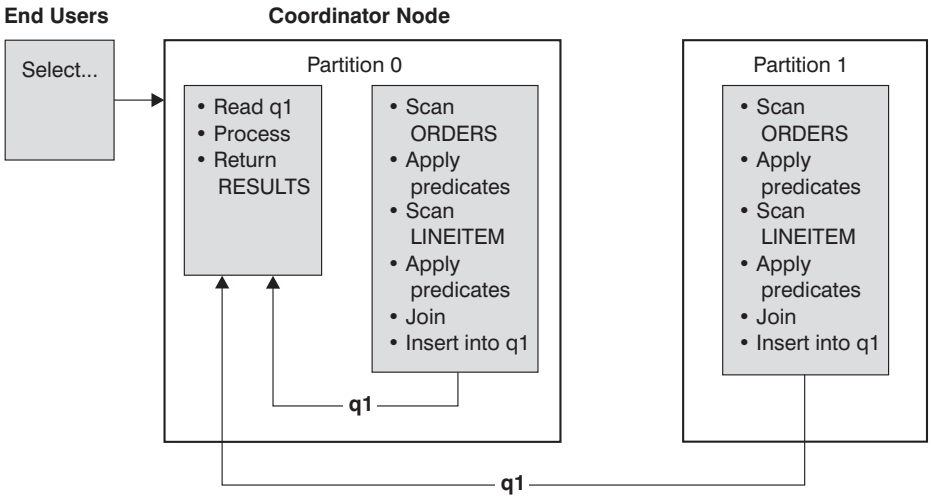
Note: In the diagrams q1, q2, and q3 refer to table queues in the examples. The tables that are referenced are divided across two database partitions for the purpose of these scenarios. The arrows indicate the direction in which the table queues are sent. The coordinator node is database partition 0.

Collocated Joins

A collocated join occurs locally on the database partition where the data resides. The database partition sends the data to the other database partitions after the join is complete. For the optimizer to consider a collocated join, the joined tables must be collocated, and all pairs of the corresponding distribution key must participate in the equality join predicates.

The following figure provides an example.

Note: Replicated materialized query tables enhance the likelihood of collocated joins.

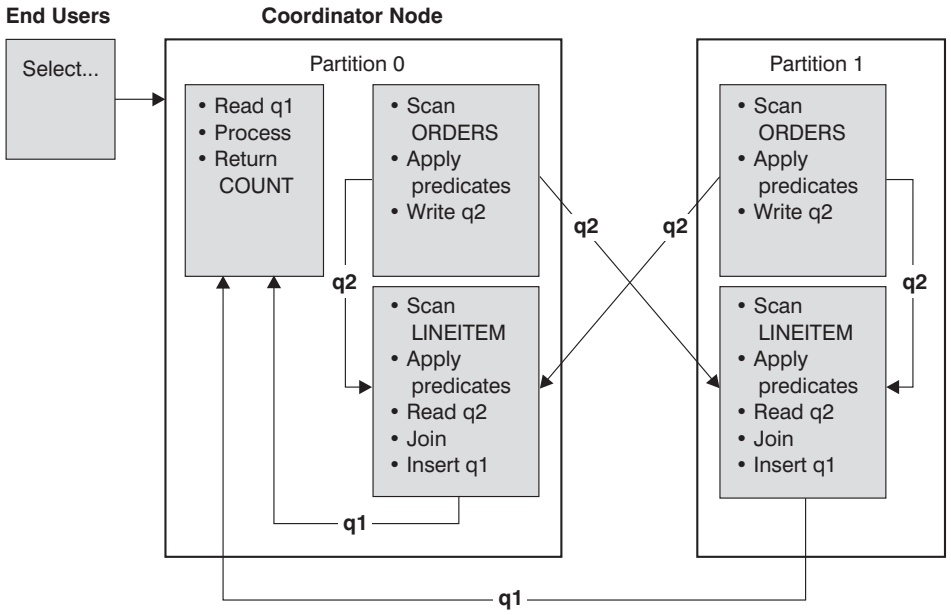


Both the LINEITEM and ORDERS tables are partitioned on the ORDERKEY column. The join is done locally at each database partition. In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 20. Collocated Join Example

Broadcast Outer-Table Joins

Broadcast outer-table joins are a parallel join strategy that can be used if there are no equality join predicates between the joined tables. It can also be used in other situations in which it is the most cost-effective join method. For example, a broadcast outer-table join might occur when there is one very large table and one very small table, neither of which is split on the join predicate columns. Instead of splitting both tables, it might be cheaper to broadcast the smaller table to the larger table. The following figures provide an example.

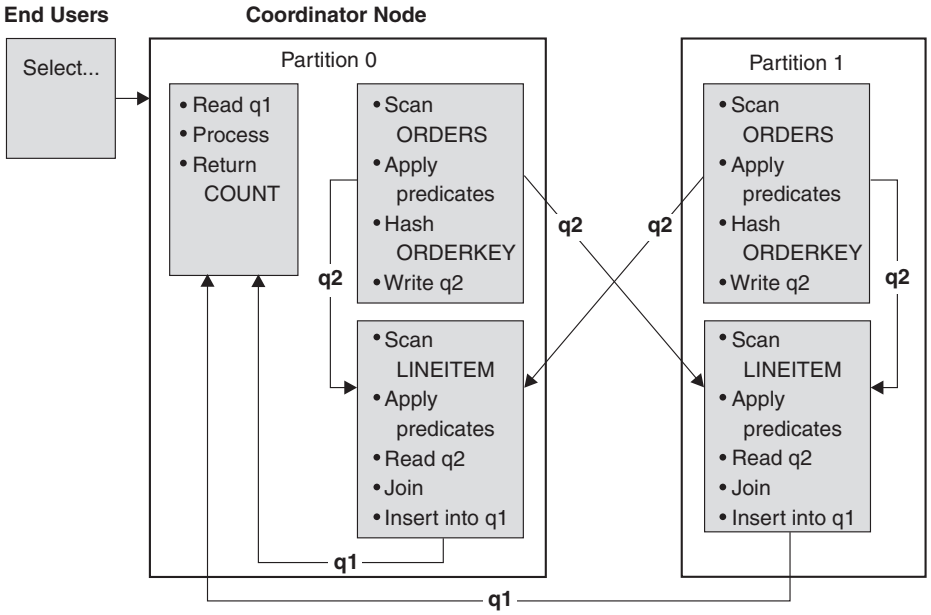


The ORDERS table is sent to all database partitions that have the LINEITEM table. Table queue q2 is broadcast to all database partitions of the inner table.

Figure 21. Broadcast Outer-Table Join Example

Directed Outer-Table Joins

In the directed outer-table join strategy, each row of the outer table is sent to one portion of the inner table, based on the splitting attributes of the inner table. The join occurs on this database partition. The following figure provides an example.

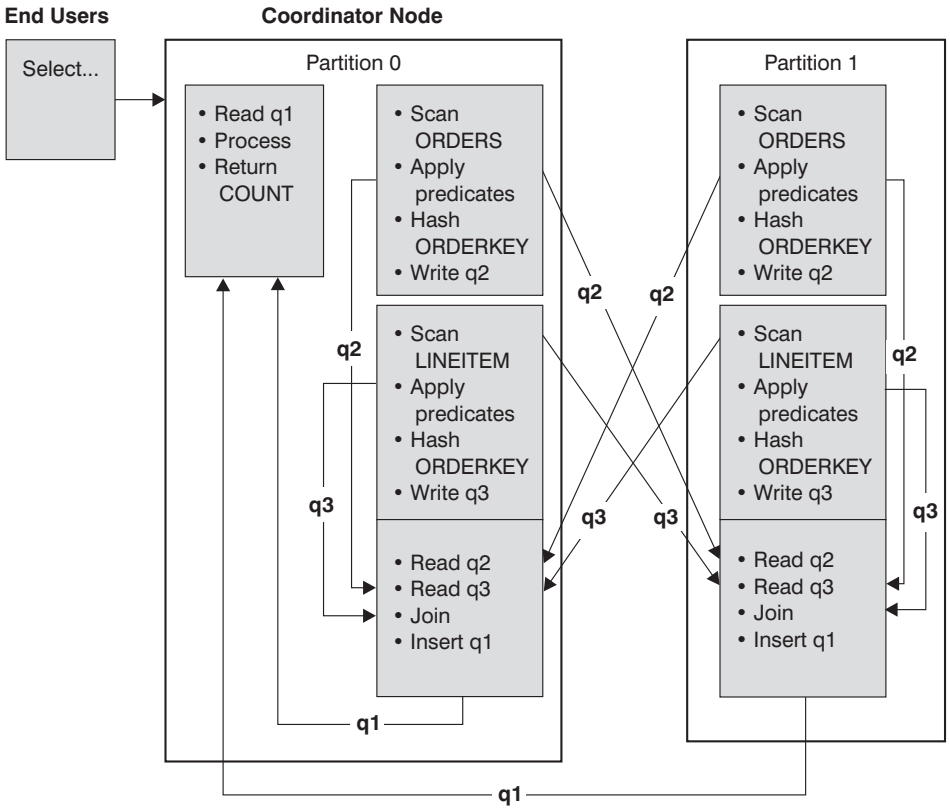


The LINEITEM table is partitioned on the ORDERKEY column.
The ORDERS table is partitioned on a different column.
The ORDERS table is hashed and sent to the correct LINEITEM table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 22. Directed Outer-Table Join Example

Directed Inner-Table and Outer-Table Joins

In the directed inner-table and outer-table join strategy, rows of both the outer and inner tables are directed to a set of database partitions, based on the values of the joining columns. The join occurs on these database partitions. The following figure provides an example. An example is shown in the following figure.

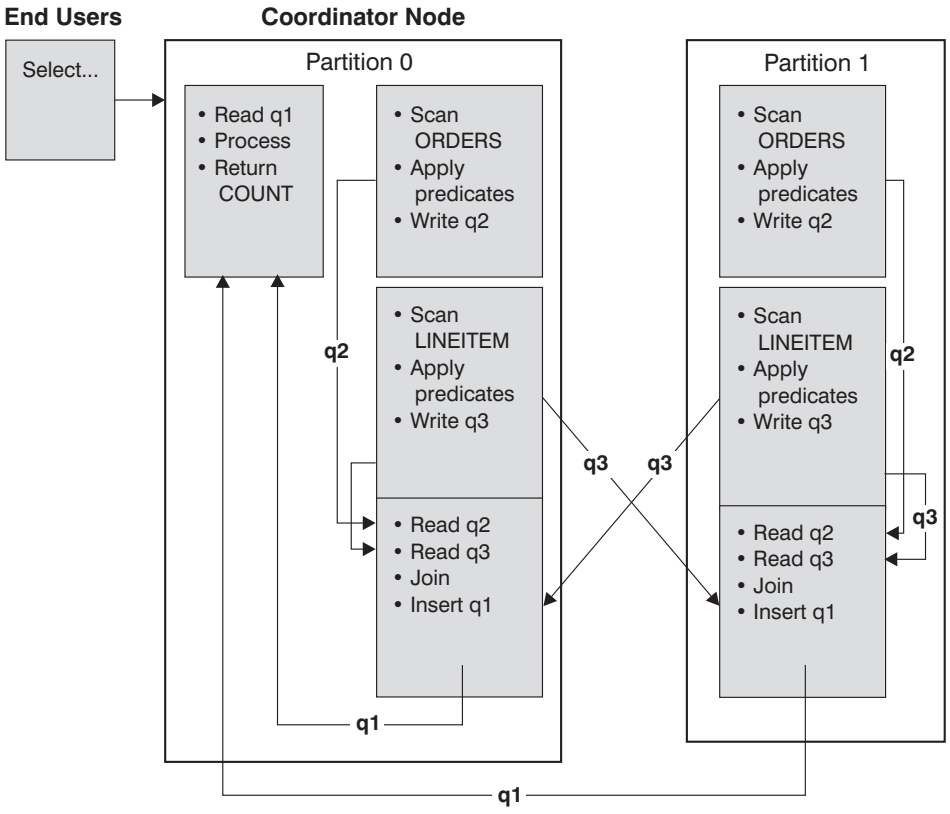


Neither table is partitioned on the ORDERKEY column.
Both tables are hashed and are sent to new database partitions where they are joined.
Both table queue q2 and q3 are directed.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY

Figure 23. Directed Inner-Table and Outer-Table Join Example

Broadcast Inner-Table Joins

In the broadcast inner-table join strategy, the inner table is broadcast to all the database partitions of the outer join table. The following figure provides an example.

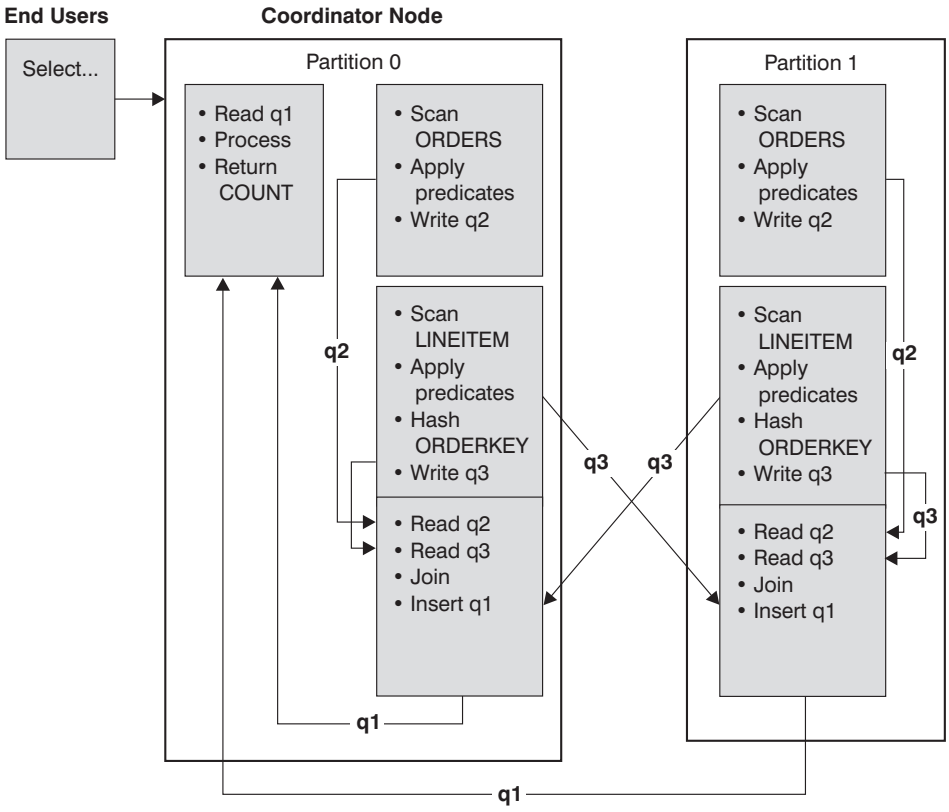


The LINEITEM table is sent to all database partitions that have the ORDERS table. Table queue q3 is broadcast to all database partitions of the outer table.

Figure 24. Broadcast Inner-Table Join Example

Directed Inner-Table Joins

With the directed inner-table join strategy, each row of the inner table is sent to one database partition of the outer join table, based on the splitting attributes of the outer table. The join occurs on this database partition. The following figure provides an example.



The ORDERS table is partitioned on the ORDERKEY column.
The LINEITEM table is partitioned on a different column.
The LINEITEM table is hashed and sent to the correct ORDERS table database partition.
In this example, the join predicate is assumed to be:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY.

Figure 25. Directed Inner-Table Join Example

Related concepts:

- "Join methods" on page 188
- "Join strategies in partitioned databases" on page 195
- "Joins" on page 187

Query Rewriting

Query rewriting methods and examples

During the rewrite query stage, the query compiler transforms SQL and XQuery statements into forms that can be optimized more easily, and as a result, can improve the possible access paths. Rewriting queries is particularly important for very complex queries, including those queries with many subqueries or many joins. Query generator tools often create these types of very complex queries.

To influence the number of query rewrite rules that are applied to an SQL or XQuery statement, change the optimization class. To see some of the results of the query rewrite, use the Explain facility or Visual Explain.

Queries might be rewritten in any of the following three primary ways:

- **Operation merging**

  To construct the query so that it has the fewest number of operations, especially SELECT operations, the SQL and XQuery compiler rewrites queries to merge query operations. The following examples illustrate some of the operations that can be merged:

  – Example - View Merges

    A SELECT statement that uses views can restrict the join order of the table and can also introduce redundant joining of tables. If the views are merged during query rewrite, these restrictions can be lifted.

  – Example - Subquery to Join Transforms

    If a SELECT statement contains a subquery, selection of order processing of the tables might be restricted.

  – Example - Redundant Join Elimination

    During query rewrite, redundant joins can be removed to simplify the SELECT statement.

  – Example - Shared Aggregation

    When the query uses different functions, rewriting can reduce the number of calculations that need to be done.

- **Operation movement**

  To construct the query with the minimum number of operations and predicates, the compiler rewrites queries to move query operations. The following examples illustrate some of the operations that can be moved:

  – Example - DISTINCT Elimination

    During query rewrite, the optimizer can move the point at which the DISTINCT operation is performed, to reduce the cost of this operation. In the extended example provided, the DISTINCT operation is removed completely.

  – Example - General Predicate Pushdown

    During query rewrite, the optimizer can change the order of applying predicates so that more selective predicates are applied to the query as early as possible.

  – Example - Decorrelation

    In a partitioned database environment, the movement of results sets between database partitions is costly. Reducing the size of what must be broadcast to other database partitions, or reducing the number of broadcasts, or both, is an objective of query rewriting.

- **Predicate Translation**

  The SQL and XQuery compiler rewrites queries to translate existing predicates to more optimal predicates for the specific query. The following examples illustrate some of the predicates that might be translated:

  – Example - Addition of Implied Predicates

    During query rewrite, predicates can be added to the query to allow the optimizer to consider additional table joins when selecting the best access plan for the query.

  – Example - OR to IN Transformations

    During query rewriting, an OR predicate can be translated into an IN predicate for a more efficient access plan. The SQL and XQuery compiler can also translate an IN predicate into an OR predicate if this transformation would create a more efficient access plan.

Related concepts:

- "Column correlation for multiple predicates" on page 193
- "Compiler rewrite example: DISTINCT elimination" on page 207
- "Compiler rewrite example: implied predicates" on page 209
- "Compiler rewrite example: view merges" on page 205

Compiler rewrite example: view merges

Suppose you have access to the following two views of the EMPLOYEE table, one showing employees with a high level of education and the other view showing employees earning more than $35,000:

```
CREATE VIEW EMP_EDUCATION (EMPNO, FIRSTNME, LASTNAME, EDLEVEL) AS
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL
  FROM EMPLOYEE
 WHERE EDLEVEL > 17
CREATE VIEW EMP_SALARIES (EMPNO, FIRSTNAME, LASTNAME, SALARY) AS
SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
  FROM EMPLOYEE
 WHERE SALARY > 35000
```

Now suppose you perform the following query to list the employees who have a high education level and who are earning more than $35,000:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMP_EDUCATION E1,
       EMP_SALARIES  E2
 WHERE E1.EMPNO = E2.EMPNO
```

During query rewrite, these two views could be merged to create the following query:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

By merging the SELECT statements from the two views with the user-written SELECT statement, the optimizer can consider more choices when selecting an access plan. In addition, if the two views that have been merged use the same base table, additional rewriting may be performed.

Example - Subquery to Join Transformations

The SQL and XQuery compiler will take a query containing a subquery, such as:

```
SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO
FROM EMPLOYEE
WHERE WORKDEPT IN
      (SELECT DEPTNO
         FROM DEPARTMENT
        WHERE DEPTNAME = 'OPERATIONS')
```

and convert it to a join query of the form:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME, PHONENO
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNAME = 'OPERATIONS'
```

A join is generally much more efficient to execute than a subquery.

Example - Redundant Join Elimination

Queries can sometimes be written or generated which have unnecessary joins. Queries such as the following could also be produced during the query rewrite stage.

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
  FROM EMPLOYEE E1,
       EMPLOYEE E2
 WHERE E1.EMPNO = E2.EMPNO
   AND E1.EDLEVEL > 17
   AND E2.SALARY  > 35000
```

In this query, the SQL and XQuery compiler can eliminate the join and simplify the query to:

```
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL, SALARY
  FROM EMPLOYEE
 WHERE EDLEVEL > 17
   AND SALARY  > 35000
```

Another example assumes that a referential constraint exists between the EMPLOYEE and DEPARTMENT sample tables on the department number. First, a view is created.

```
CREATE VIEW PEPLVIEW
   AS SELECT FIRSTNME, LASTNAME, SALARY, DEPTNO, DEPTNAME, MGRNO
         FROM EMPLOYEE E DEPARTMENT D
         WHERE E.WORKDEPT = D.DEPTNO
```

Then a query such as the following:

```
SELECT LASTNAME, SALARY
   FROM PEPLVIEW
```

becomes

```
SELECT LASTNAME, SALARY
   FROM EMPLOYEE
   WHERE WORKDEPT NOT NULL
```

Note that in this situation, even if users know that the query can be re-written, they may not be able to do so because they do not have access to the underlying tables. They may only have access to the view shown above. Therefore, this type of optimization has to be performed within the database manager.

Redundancy in referential integrity joins is likely where:

- Views are defined with joins
- Queries are automatically generated.

  For example, there are automated tools in query managers which prevent users from writing optimized queries.

Example - Shared Aggregation

Using multiple functions within a query can generate several calculations which take time. Reducing the number of calculations to be done within the query results in an improved plan. The SQL and XQuery compiler takes a query using multiple functions such as:

```
SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  AVG(SALARY+BONUS+COMM) AS OAVG,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE;
```

and transforms the query in the following way:

```
SELECT OSUM,
  OSUM/OCOUNT
  OCOUNT
FROM (SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
  COUNT(*) AS OCOUNT
FROM EMPLOYEE) AS SHARED_AGG;
```

This rewrite reduces the query from 2 sums and 2 counts to 1 sum and 1 count.

Related concepts:

- "Query rewriting methods and examples" on page 203
- "The SQL and XQuery compiler process" on page 149

Compiler rewrite example: DISTINCT elimination

If the EMPNO column was defined as the primary key of the EMPLOYEE table, the following query:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

would be rewritten by removing the DISTINCT clause:

```
SELECT EMPNO, FIRSTNME, LASTNAME
  FROM EMPLOYEE
```

In the above example, since the primary key is being selected, the SQL and XQuery compiler knows that each row returned will already be unique. In this case, the DISTINCT key word is redundant. If the query is not rewritten, the optimizer would need to build a plan with the necessary processing, such as a sort, to ensure that the columns are distinct.

Example - General Predicate Pushdown

Altering the level at which a predicate is normally applied can result in improved performance. For example, given the following view which provides a list of all employees in department "D11":

```
CREATE VIEW D11_EMPLOYEE
 (EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM)
AS SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM
     FROM EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

And given the following query:

```
SELECT FIRSTNME, PHONENO
  FROM D11_EMPLOYEE
 WHERE LASTNAME = 'BROWN'
```

The query rewrite stage of the compiler will push the predicate `LASTNAME = 'BROWN'` down into the view D11_EMPLOYEE. This allows the predicate to be applied sooner and potentially more efficiently. The actual query that could be executed in this example is:

```
SELECT FIRSTNME, PHONENO
  FROM EMPLOYEE
 WHERE LASTNAME = 'BROWN'
   AND WORKDEPT = 'D11'
```

Pushdown of predicates is not limited to views. Other situations in which predicates may be pushed down include UNIONs, GROUP BYs, and derived tables (nested table expressions or common table expressions).

Example - Decorrelation

In a partitioned database environment, the SQL and XQuery compiler can rewrite the following query:

Find all the employees who are working on programming projects and are underpaid.

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
  FROM EMPLOYEE E, PROJECT P
 WHERE P.EMPNO = E.EMPNO
   AND P.PROJNAME LIKE '%PROGRAMMING%'
   AND E.SALARY+E.BONUS+E.COMM <
     (SELECT AVG(E1.SALARY+E1.BONUS+E1.COMM)
        FROM EMPLOYEE E1, PROJECT P1
        WHERE P1.PROJNAME LIKE '%PROGRAMMING%'
          AND P1.PROJNO = A.PROJNO
          AND E1.EMPNO = P1.EMPNO)
```

Since this query is correlated, and since both PROJECT and EMPLOYEE are unlikely to be partitioned on PROJNO, the broadcast of each project to each database partition is possible. In addition, the subquery would have to be evaluated many times.

The SQL and XQuery compiler can rewrite the query as follows:

- Determine the distinct list of employees working on programming projects and call it DIST_PROJS. It must be distinct to ensure that aggregation is done once only for each project:

  ```
  WITH DIST_PROJS(PROJNO, EMPNO) AS
  (SELECT DISTINCT PROJNO, EMPNO
   FROM PROJECT P1
   WHERE P1.PROJNAME LIKE '%PROGRAMMING%')
  ```

- Using the distinct list of employees working on the programming projects, join this to the employee table, to get the average compensation per project, AVG_PER_PROJ:

  ```
  AVG_PER_PROJ(PROJNO, AVG_COMP) AS
  (SELECT P2.PROJNO, AVG(E1.SALARY+E1.BONUS+E1.COMM)
   FROM EMPLOYEE E1, DIST_PROJS P2
   WHERE E1.EMPNO = P2.EMPNO
   GROUP BY P2.PROJNO)
  ```

- Then the new query would be:

  ```
  SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
         E.SALARY+E.BONUS+E.COMM AS COMPENSATION
   FROM PROJECT P, EMPLOYEE E, AVG_PER_PROG A
  WHERE P.EMPNO = E.EMPNO
    AND P.PROJNAME LIKE '%PROGRAMMING%'
    AND P.PROJNO = A.PROJNO
    AND E.SALARY+E.BONUS+E.COMM < A.AVG_COMP
  ```

The rewritten query computes the AVG_COMP per project (AVG_PRE_PROJ) and can then broadcast the result to all database partitions containing the EMPLOYEE table.

Related concepts:

- "Query rewriting methods and examples" on page 203
- "The SQL and XQuery compiler process" on page 149

Compiler rewrite example: implied predicates

The following query produces a list of the managers whose departments report to "E01" and the projects for which those managers are responsible:

```
SELECT DEPT.DEPTNAME DEPT.MGRNO, EMP.LASTNAME, PROJ.PROJNAME
  FROM DEPARTMENT DEPT,
       EMPLOYEE   EMP,
       PROJECT    PROJ
 WHERE DEPT.ADMRDEPT = 'E01'
   AND DEPT.MGRNO = EMP.EMPNO
   AND EMP.EMPNO  = PROJ.RESPEMP
```

The query rewrite adds the following implied predicate:

```
DEPT.MGRNO = PROJ.RESPEMP
```

As a result of this rewrite, the optimizer can consider additional joins when it is trying to select the best access plan for the query.

In addition to the above predicate transitive closure, query rewrite also derives additional local predicates based on the transitivity implied by equality predicates. For example, the following query lists the names of the departments whose department number is greater than "E00" and the employees who work in those departments.

```
SELECT EMPNO, LASTNAME, FIRSTNAME, DEPTNO, DEPTNAME
  FROM EMPLOYEE EMP,
       DEPARTMENT DEPT
 WHERE EMP.WORKDEPT = DEPT.DEPTNO
   AND DEPT.DEPTNO > 'E00'
```

For this query, the rewrite stage adds the following implied predicate:

```
EMP.WORKDEPT > 'E00'
```

As a result of this rewrite, the optimizer reduces the number of rows to be joined.

Example - OR to IN Transformations

Suppose an OR clause connects two or more simple equality predicates on the same column, as in the following example:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO = 'D11'
    OR DEPTNO = 'D21'
    OR DEPTNO = 'E21'
```

If there is no index on the DEPTNO column, converting the OR clause to the following IN predicate allows the query to be processed more efficiently:

```
SELECT *
  FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

Note: In some cases, the database manager might convert an IN predicate to a set of OR clauses so that index ORing might be performed.

Related concepts:

- "Query rewriting methods and examples" on page 203
- "The SQL and XQuery compiler process" on page 149

Access Plans in Federated Queries

Federated database pushdown analysis

For queries in federated databases, the optimizer performs pushdown analysis to find out whether an operation can be performed at a remote data source. An operation might be a function, such as relational operator, system or user function, or an SQL operator, such as GROUP BY, ORDER BY, and so on.

Note: Although the DB2 SQL compiler has much information about data source SQL support, this data may need adjustment over time because data sources can be upgraded and/or customized. In such cases, make enhancements known to DB2 by changing local catalog information. Use DB2 DDL statements (such as CREATE FUNCTION MAPPING and ALTER SERVER) to update the catalog.

If functions cannot be pushed down to the remote data source, they can significantly impact query performance. Consider the effect of forcing a selective predicate to be evaluated locally instead of at the data source. Such evaluation could require DB2 to retrieve the entire table from the remote data source and then filter it locally against the predicate. Network constraints and large table size could cause performance to suffer.

Operators that are not pushed down can also significantly affect query performance. For example, having a GROUP BY operator aggregate remote data locally could also require DB2 to retrieve the entire table from the remote data source.

For example, assume that a nickname N1 references the data source table EMPLOYEE in a DB2 for OS/390 or z/OS data source. Also assume that the table has 10,000 rows, that one of the columns contains the last names of employees, and that one of the columns contains salaries. Consider the following statement:

```
SELECT LASTNAME, COUNT(*)  FROM N1
   WHERE LASTNAME  > 'B' AND SALARY > 50000
   GROUP BY LASTNAME;
```

Several possibilities are considered, depending on whether the collating sequences at DB2 and DB2 for OS/390 or z/OS are the same:

- If the collating sequences are the same, the query predicate can probably be pushed down to DB2 for OS/390 or z/OS. Filtering and grouping results at the data source is usually more efficient than copying the entire table to DB2 and performing the operations locally. For the query above, the predicate and the GROUP BY operation can take place at the data source.
- If the collating sequence is not the same, the entire predicate cannot be evaluated at the data source. However, the optimizer might decide to pushdown the `SALARY > 50000` portion of the predicate. The range comparison must still be done at DB2.
- If the collating sequence is the same, and the optimizer knows that the local DB2 server is very fast, the optimizer might decide that performing the `GROUP BY` operation locally at DB2 is the best (least cost) approach. The predicate is evaluated at the data source. This is an example of pushdown analysis combined with global optimization.

In general, the goal is to ensure that the optimizer evaluates functions and operators on data sources. Many factors affect whether a function or an SQL operator is evaluated at a remote data source. Factors to be evaluated are classified in the following three groups:

- Server characteristics
- Nickname characteristics
- Query characteristics

Server characteristics that affect pushdown opportunities

Certain data source-specific factors can affect pushdown opportunities. In general, these factors exist because of the rich SQL dialect supported by DB2. This dialect might offer more functionality than the SQL dialect supported by a server accessed by a query. DB2 can compensate for the lack of function at a data server, but doing so may require that the operation take place at DB2.

SQL Capabilities: Each data source supports a variation of the SQL dialect and different levels of functionality. For example, consider the GROUP BY list. Most data sources support the GROUP BY operator, but some limit the number of items on the GROUP BY list or have restrictions on whether an expression is allowed on the GROUP BY list. If there is a restriction at the remote data source, DB2 might have to perform the GROUP BY operation locally.

SQL Restrictions: Each data source might have different SQL restrictions. For example, some data sources require parameter markers to bind values to remote SQL statements. Therefore, parameter marker restrictions must be checked to ensure that each data source can support such a bind mechanism. If DB2 cannot determine a good method to bind a value for a function, this function must be evaluated locally.

SQL Limits: Although DB2 might allow the use of larger integers than its remote data sources, values that exceed remote limits cannot be embedded in statements sent to data sources. Therefore, the function or operator that operates on this constant must be evaluated locally.

Server Specifics: Several factors fall into this category. One example is whether NULL values are sorted as the highest or lowest value, or depend on the ordering. If NULL values are sorted at a data source differently from DB2, ORDER BY operations on a nullable expression cannot be remotely evaluated.

Collating Sequence: Retrieving data for local sorts and comparisons usually decreases performance. Therefore, consider configuring the federated database to use the same collating sequences that your data sources use. If you configure a federated database to use the same collating sequence that a data source uses and then set the *collating_sequence* server option to 'Y', the optimizer can consider pushing down many query operations if improved performance results.

The following operations might be pushed down if collating sequences are the same:

- Comparisons of character or numeric data
- Character range comparison predicates
- Sorts

You might get unusual results, however, if the weighting of null characters is different between the federated database and the data source. Comparison statements might return unexpected results if you submit statements to a case-insensitive data source. The weights assigned to the characters ″I″ and ″i″ in a case-insensitive data source are the same. DB2, by default, is case sensitive and assigns different weights to the characters.

To improve performance, the federated server allows sorts and comparisons to take place at data sources. For example, in DB2 for OS/390 or z/OS, sorts defined by ORDER BY clauses are implemented by a collating sequence based on an EBCDIC code page. To use the federated server to retrieve DB2 for OS/390 or z/OS data sorted in accordance with ORDER BY clauses, configure the federated database so that it uses a predefined collating sequence based on the EBCDIC code page.

If the collating sequences of the federated database and the data source differ, DB2 retrieves the data to the federated database. Because users expect to see the query results ordered by the collating sequence defined for the federated server, by ordering the data locally the federated server ensures that this expectation is fulfilled. Submit your query in pass-through mode, or define the query in a data source view if you need to see the data ordered in the collating sequence of the data source.

Server Options: Several server options can affect pushdown opportunities. In particular, review your settings for *collating_sequence*, *varchar_no_trailing_blanks*, and *pushdown*.

DB2 Type Mapping and Function Mapping Factors: The default local data type mappings provided by DB2 are designed to provide sufficient buffer space for each data source data type, which avoids loss of data. Users can customize the type mapping for a specific data source to suit specific applications. For example, if you are accessing an Oracle data source column with a DATE data type, which by default is mapped to the DB2 TIMESTAMP data type, you might change the local data type to the DB2 DATE data type.

In the following three cases, DB2 can compensate for functions that a data source does not support:

- The function does not exist at the remote data source.
- The function exists, but the characteristics of the operand violate function restrictions. An example of this situation is the IS NULL relational operator. Most data sources support it, but some may have restrictions, such as only allowing a column name on the left hand side of the IS NULL operator.
- The function might return a different result if it is evaluated remotely. An example of this situation is the '>' (greater than) operator. For data sources with different collating sequences, the greater than operator might return different results than if it is evaluated locally by DB2.

Nickname characteristics that affect pushdown opportunities

The following nickname-specific factors can affect pushdown opportunities.

Local data type of a nickname column: Ensure that the local data type of a column does not prevent a predicate from being evaluated at the data source. Use the default data type mappings to avoid possible overflow. However, a joining predicate between two columns of different lengths might not be considered at the data source whose joining column is shorter, depending on how DB2 binds the

longer column. This situation can affect the number of possibilities that the DB2 optimizer can evaluate in a joining sequence. For example, Oracle data source columns created using the INTEGER or INT data type are given the type NUMBER(38). A nickname column for this Oracle data type is given the local data type FLOAT because the range of a DB2 integer is from 2**31 to (-2**31)-1, which is roughly equal to NUMBER(9). In this case, joins between a DB2 integer column and an Oracle integer column cannot take place at the DB2 data source (shorter joining column); however, if the domain of this Oracle integer column can be accommodated by the DB2 INTEGER data type, change its local data type with the ALTER NICKNAME statement so that the join can take place at the DB2 data source.

Column Options: Use the SQL statement ALTER NICKNAME to add or change column options for nicknames.

Use the *varchar_no_trailing_blanks* option to identify a column that contains no trailing blanks. The compiler pushdown analysis step will then take this information into account when checking all operations performed on columns so indicated. Based on this indication, DB2 may generate a different but equivalent form of a predicate to be used in the remote SQL statement sent to a data source. A user might see a different predicate being evaluated against the data source, but the net result should be equivalent.

Use the *numeric_string* option to indicate whether the values in that column are always numbers without trailing blanks.

The table below describes these options.

Table 59. Column Options and Their Settings

Option	Valid Settings		Default Setting
numeric_string	'Y'	Yes, this column contains only strings of numeric data. IMPORTANT: If the column contains only numeric strings followed by trailing blanks, do not specify 'Y'.	'N'
	'N'	No, this column is not limited to strings of numeric data.	
	If you set numeric_string to 'Y' for a column, you are informing the optimizer that this column contains no blanks that could interfere with sorting of the column data. This option is useful when the collating sequence of a data source is different from DB2. Columns marked with this option are not excluded from local (data source) evaluation because of a different collating sequence.		
varchar_no_trailing_blanks	Specifies whether this data source uses non-blank padded VARCHAR comparison semantics. For variable-length character strings that contain no trailing blanks, non-blank-padded comparison semantics of some DBMSs return the same results as DB2 comparison semantics. If you are certain that all VARCHAR table/view columns at a data source contain no trailing blanks, consider setting this server option to 'Y' for a data source. This option is often used with Oracle data sources. Ensure that you consider all objects that might have nicknames, including views.		'N'
	'Y'	This data source has non-blank-padded comparison semantics similar to DB2.	
	'N'	This data source does not have the same non-blank-padded comparison semantics as DB2.	

Query characteristics that affect pushdown opportunities

A query can reference an SQL operator that might involve nicknames from multiple data sources. The operation must take place at DB2 to combine the results from two referenced data sources that use one operator, such as a set operator (e.g. UNION). The operator cannot be evaluated at a remote data source directly.

Related concepts:

- "Guidelines for analyzing where a federated query is evaluated" on page 214

Guidelines for analyzing where a federated query is evaluated

DB2 provides two utilities to show where queries are evaluated:

- Visual explain. Start it with the **db2cc** command. Use it to view the query access plan graph. The execution location for each operator is included in the detailed display of an operator.

  If a query is pushed down, you should see a RETURN operator. The RETURN operator is a standard DB2 operator. For a SELECT statement that selects data from a nickname, you also see a SHIP operator. The SHIP operator is unique to federated database operations. It changes the server property of the data flow and separates local operators from remote operators. The SELECT statement is generated using the SQL dialect supported by the data source. It can contain any valid query for that data source.

  If an INSERT, DELETE, or UPDATE query can be entirely pushed down to the remote database, you might not see a SHIP statement in the access plan. All remotely executed INSERT, UPDATE, and DELETE statements are shown for the RETURN operator. However, if a query cannot be entirely pushed down, the SHIP operator shows which operations were performed remotely.
- SQL explain. Start it with the **db2expln** or the **dynexpln** command. Use it to view the access plan strategy as text.

Understanding why a query is evaluated at a data source or at DB2

Consider the following key questions when you investigate ways to increase pushdown opportunities:

- Why isn't this predicate being evaluated remotely?

  This question arises when a predicate is very selective and thus could be used to filter rows and reduce network traffic. Remote predicate evaluation also affects whether a join between two tables of the same data source can be evaluated remotely.

  Areas to examine include:

  – Subquery predicates. Does this predicate contain a subquery that pertains to another data source? Does this predicate contain a subquery involving an SQL operator that is not supported by this data source? Not all data sources support set operators in a subquery predicate.

  – Predicate functions. Does this predicate contain a function that cannot be evaluated by this remote data source? Relational operators are classified as functions.

  – Predicate bind requirements. Does this predicate, if remotely evaluated, require bind-in of some value? If so, would it violate SQL restrictions at this data source?

  – Global optimization. The optimizer may have decided that local processing is more cost effective.

- Why isn't the GROUP BY operator evaluated remotely?

  There are several areas you can check:

  – Is the input to the GROUP BY operator evaluated remotely? If the answer is no, examine the input.
  – Does the data source have any restrictions on this operator? Examples include:
    - Limited number of GROUP BY items
    - Limited byte counts of combined GROUP BY items
    - Column specification only on the GROUP BY list
  – Does the data source support this SQL operator?
  – Global optimization. The optimizer may have decided that local processing is more cost effective.
  – Does the GROUP BY operator clause contain a character expression? If it does, verify that the remote data source has the same case sensitivity as DB2.
- Why isn't the set operator evaluated remotely?

  There are several areas you can check:

  – Are both of its operands completely evaluated at the same remote data source? If the answer is no and it should be yes, examine each operand.
  – Does the data source have any restrictions on this set operator? For example, are large objects or long fields valid input for this specific set operator?
- Why isn't the ORDER BY operation evaluated remotely?

  Consider:

  – Is the input to the ORDER BY operation evaluated remotely? If the answer is no, examine the input.
  – Does the ORDER BY clause contain a character expression? If yes, does the remote data source not have the same collating sequence or case sensitivity as DB2?
  – Does the data source have any restrictions on this operator? For example, is there a limited number of ORDER BY items? Does the data source restrict column specification to the ORDER BY list?

Related concepts:

- "Federated query information" on page 649
- "Global analysis of federated database queries" on page 215
- "Character-conversion guidelines" in *Administration Guide: Planning*
- "Remote SQL generation and global optimization in federated databases" on page 217

Global analysis of federated database queries

The following two utilities provided show global access plans:

- Visual Explain. Start it from the Control Center, or execute the *db2cc* command, which starts the Control Center. Use Visual Explain to view the query access plan graph. The execution location for each operator is included in the detailed display of an operator. You can also find the remote SQL statement generated for each data source in the SHIP or RETURN operator, depending on the type of the query. By examining the details of each operator, you can see the number of rows estimated by the DB2 optimizer as input to and output from each operator. You can also see the estimated cost to execute each operator including the communications cost.

- SQL explain. Start it with the *db2expln* or *dynexpln* command. Use SQL explain to view the access plan strategy as text. Although SQL explain does not provide cost information, you can get the access plan generated by the remote optimizer for those data sources supported by the remote explain function.

Understanding DB2 optimization decisions

Consider the following optimization questions and key areas to investigate for performance improvements:

- Why isn't a join between two nicknames of the same data source being evaluated remotely?

  Areas to examine include:

  – Join operations. Can the data source support them?

  – Join predicates. Can the join predicate be evaluated at the remote data source? If the answer is no, examine the join predicate.

  – Number of rows in the join result (with Visual Explain). Does the join produce a much larger set of rows than the two nicknames combined? Do the numbers make sense? If the answer is no, consider updating the nickname statistics manually (SYSSTAT.TABLES).

- Why isn't the GROUP BY operator being evaluated remotely?

  Areas to examine include:

  – Operator syntax. Verify that the operator can be evaluated at the remote data source.

  – Number of rows. Check the estimated number of rows in the GROUP BY operator input and output using visual explain. Are these two numbers very close? If the answer is yes, the DB2 optimizer considers it more efficient to evaluate this GROUP BY locally. Also, do these two numbers make sense? If the answer is no, consider updating the nickname statistics manually (SYSSTAT.TABLES).

- Why is the statement not being completely evaluated by the remote data source?

  The DB2 optimizer performs cost-based optimization. Even if pushdown analysis indicates that every operator can be evaluated at the remote data source, the optimizer still relies on its cost estimate to generate a globally optimal plan. There are a great many factors that can contribute to that plan. For example, even though the remote data source can process every operation in the original query, its CPU speed is much slower than the CPU speed for DB2 and thus it may turn out to be more beneficial to perform the operations at DB2 instead. If results are not satisfactory, verify your server statistics in SYSCAT.SERVEROPTIONS.

- Why does a plan generated by the optimizer, and completely evaluated at a remote data source, have much worse performance than the original query executed directly at the remote data source?

  Areas to examine include:

  – The remote SQL statement generated by the DB2 optimizer. Ensure that it is identical to the original query. Check for predicate ordering changes. A good query optimizer should not be sensitive to the predicate ordering of a query; unfortunately, not all DBMS optimizers are identical, and thus it is likely that the optimizer of the remote data source may generate a different plan based on the input predicate ordering. If this is true, this is a problem inherent in the remote optimizer. Consider either modifying the predicate ordering on the input to DB2 or contacting the service organization of the remote data source for assistance.

Also, check for predicate replacements. A good query optimizer should not be sensitive to equivalent predicate replacements; unfortunately, not all DBMS optimizers are identical, and thus it is possible that the optimizer of the remote data source may generate a different plan based on the input predicate. For example, some optimizers cannot generate transitive closure statements for predicates.

– The number of returned rows. You can get this number from Visual Explain. If the query returns a large number of rows, network traffic is a potential bottleneck.
– Additional functions. Does the remote SQL statement contain additional functions compared with the original query? Some of the extra functions may be generated to convert data types. Ensure that they are necessary.

Related concepts:

- "Example five: federated database plan" on page 660
- "Federated database pushdown analysis" on page 210
- "Federated query information" on page 649
- "Guidelines for analyzing where a federated query is evaluated" on page 214
- "Remote SQL generation and global optimization in federated databases" on page 217
- "Server options affecting federated databases" on page 219

Remote SQL generation and global optimization in federated databases

For a federated database query that uses relational nicknames, the access strategy might involve breaking down the original query into a set of remote query units and then combining the results. This generation of remote SQL helps produce a globally optimal access strategy for a query.

The optimizer uses the output of pushdown analysis to decide whether each operation is evaluated locally at DB2 or remotely at a data source. It bases its decision on the output of its cost model, which includes not only the cost of evaluating the operation but also the cost of transmitting the data or messages between DB2 and data sources.

Although the goal is to produce an optimized query, the following major factors affect the output from global optimization and thus affect query performance.

- Server characteristics
- Nickname characteristics

Server characteristics and options that affect global optimization

The following data source server factors can affect global optimization:

- Relative ratio of CPU speed

  Use the *cpu_ratio* server option to specify how fast or slow the data-source CPU speed is compared with the DB2 CPU. A low ratio indicates that the data-source computer CPU is faster than the DB2 computer CPU. If the ratio is low, the DB2 optimizer is more likely to consider pushing down CPU-intensive operations to the data source.
- Relative ratio of I/O speed

Use the *io_ratio* server option to indicate how much faster or slower the data source system I/O speed is compared with the DB2 system. A low ratio indicates that the data source workstation I/O speed is faster than the DB2 workstation I/O speed. If the ratio is low, the DB2 optimizer considers pushing down I/O-intensive operations to the data source.

- Communication rate between DB2 and the data source

  Use the *comm_rate* server option to indicate network capacity. Low rates, which indicate a slow network communication between DB2 and the data source, encourage the DB2 optimizer to reduce the number of messages sent to or from this data source. If the rate is set to 0, the optimizer creates an access plan that requires minimal network traffic.

- Data source collating sequence

  Use the *collating_sequence* server option to specify whether a data source collating sequence matches the local DB2 database collating sequence. If this option is not set to 'Y', the optimizer considers the data retrieved from this data source as unordered.

- Remote plan hints

  Use the *plan_hints* server option to specify that plan hints should be generated or used at a data source. By default, DB2 does not send any plan hints to the data source.

  Plan hints are statement fragments that provide extra information for data-source optimizers. For some queries this information can improve performance. The plan hints can help the data source optimizer decide whether to use an index, which index to use, or which table join sequence to use.

  If plan hints are enabled, the query sent to the data source contains additional information. For example, a statement sent to an Oracle optimizer with plan hints might look like this:

  ```
  SELECT /*+ INDEX (table1, t1index)*/
     col1
     FROM table1
  ```

  The plan hint is the string `/*+ INDEX (table1, t1index)*/`.

- Information in the DB2 optimizer knowledge base

  DB2 has an optimizer knowledge base that contains data about native data sources. The DB2 optimizer does not generate remote access plans that cannot be generated by specific DBMSs. In other words, DB2 avoids generating plans that optimizers at remote data sources cannot understand or accept.

Nickname characteristics that affect global optimization

The following nickname-specific factors can affect global optimization.

Index considerations: To optimize queries, DB2 can use information about indexes at data sources. For this reason, it is important that the index information available to DB2 is current. The index information for nicknames is initially acquired when the nickname is created. Index information is not gathered for view nicknames.

Creating index specifications on nicknames: You can create an index specification for a nickname. Index specifications build an index definition (not an actual index) in the catalog for the DB2 optimizer to use. Use the CREATE INDEX SPECIFICATION ONLY statement to create index specifications. The syntax for creating an index specification on a nickname is similar to the syntax for creating an index on a local table.

Consider creating index specifications in the following circumstances:

- DB2 cannot retrieve any index information from a data source during nickname creation.
- You want an index for a view nickname.
- You want to encourage the DB2 optimizer to use a specific nickname as the inner table of a nested loop join. The user can create an index on the joining column if none exists.

Before you issue CREATE INDEX statements against a nickname for a view, consider whether you need one. If the view is a simple SELECT on a table with an index, creating local indexes on the nickname to match the indexes on the table at the data source can significantly improve query performance. However, if indexes are created locally over views that are not simple select statements, such as a view created by joining two tables, query performance might suffer. For example, you create an index over a view that is a join of two tables, the optimizer might choose that view as the inner element in a nested-loop join. The query will have poor performance because the join is evaluated several times. An alternative is to create nicknames for each of the tables referenced in the data source view and create a local view at DB2 that references both nicknames.

Catalog statistics considerations: System catalog statistics describe the overall size of nicknames and the range of values in associated columns. The optimizer uses these statistics when it calculates the least-cost path for processing queries that contain nicknames. Nickname statistics are stored in the same catalog views as table statistics.

Although DB2 can retrieve the statistical data stored at a data source, it cannot automatically detect updates to existing statistical data at data sources. Furthermore, DB2 cannot handle changes in object definition or structural changes, such as adding a column, to objects at data sources. If the statistical data or structural data for an object has changed, you have two choices:

- Run the equivalent of RUNSTATS at the data source. Then drop the current nickname and re-create it. Use this approach if structural information has changed.
- Manually update the statistics in the SYSSTAT.TABLES view. This approach requires fewer steps but it does not work if structural information has changed.

Related concepts:

- “Global analysis of federated database queries” on page 215
- “Guidelines for analyzing where a federated query is evaluated” on page 214
- “Server options affecting federated databases” on page 219

Server options affecting federated databases

A federated system is composed of a DB2 DBMS (the federated database) and one or more data sources. You identify the data sources to the federated database when you issue CREATE SERVER statements. When you issue these statements, you can include server options that refine and control aspects of federated system operations involving DB2 and the specified data source. To change server options later, use ALTER SERVER statements.

Note: You must install the distributed join installation option and set the database manager parameter *federated* to YES before you can create servers and specify server options.

The server option values that you specify affect query pushdown analysis, global optimization and other aspects of federated database operations. For example, in the CREATE SERVER statement, you can specify performance statistics as server option values, such as the *cpu_ratio* option, which specifies the relative speeds of the CPUs at the data source and the federated server. You might also set the *io_ratio* option to a value that indicates the relative rates of the data I/O divides at the source and the federated server. When you execute the CREATE SERVER statement, this data is added to the catalog view SYSCAT.SERVEROPTIONS, and the optimizer uses it in developing its access plan for the data source. If a statistic changes (as might happen, for instance, if the data source CPU is upgraded), use the ALTER SERVER statement to update SYSCAT.SERVEROPTIONS with this change. The optimizer then uses the new information the next time it chooses an access plan for the data source.

Related reference:

- "ALTER SERVER statement" in *SQL Reference, Volume 2*
- "CREATE SEQUENCE statement" in *SQL Reference, Volume 2*
- "federated - Federated database system support " on page 501
- "SYSCAT.SERVEROPTIONS catalog view" in *SQL Reference, Volume 1*

Chapter 10. Analyzing access plans using the Explain facility

Explain facility

The SQL or XQuery compiler can capture information about the access plan and environment of static or dynamic SQL and XQuery statements. The captured information helps you understand how individual SQL or XQuery statements are executed so that you can tune the statements and your database manager configuration to improve performance.

You collect and use explain data for the following reasons:

- To understand how the database manager accesses tables and indexes to satisfy your query
- To evaluate your performance-tuning actions

  When you change some aspect of the database manager, the SQL or XQuery statements, or the database, you should examine the explain data to find out how your action has changed performance.

The captured information includes:

- Sequence of operations to process the query
- Cost information
- Predicates and selectivity estimates for each predicate
- Statistics for all objects referenced in the SQL or XQuery statement at the time that the explain information is captured
- Values for the host variables, parameter markers, or special registers used to reoptimize the SQL or XQuery statement.

Before you can capture explain information, you create the relational tables in which the optimizer stores the explain information and you set the special registers that determine what kind of explain information is captured.

To display explain information, you can use either a command-line tool or Visual Explain. The tool that you use determines how you set the special registers that determine what explain data is collected. For example, if you expect to use Visual Explain only, you need only capture snapshot information. If you expect to perform detailed analysis with one of the command-line utilities or with custom SQL or XQuery statements against the explain tables, you should capture all explain information.

Related concepts:

- "Explain tools" on page 222
- "Guidelines for analyzing explain information" on page 233
- "Guidelines for capturing explain information" on page 231
- "Guidelines for using explain information" on page 223
- "SQL and XQuery Explain tools" on page 627
- "The explain tables and organization of explain information" on page 225

dynexpln

The `dynexpln` tool is still available for backward compatibility. However, you can use the **dynamic-options** of `db2expln` to perform all of the functions of `dynexpln`.

When you use the dynamic-options of `db2expln`, the statement is prepared as true dynamic SQL or XQuery statement and the generated plan is explained from the query cache. This explain-output method provides more accurate access plans than `dynexpln`, which prepares the statement as a static SQL or XQuery statement. It also allows the use of features available only in dynamic SQL and XQuery statements, such as parameter markers.

Related concepts:

- "Examples of db2expln and dynexpln output" on page 652
- "SQL and XQuery Explain tools" on page 627

Explain tools

DB2 provides a comprehensive explain facility that provides detailed information about the access plan that the optimizer chooses for an SQL or XQuery statement. The tables that store explain data are accessible on all supported platforms and contain information for both static and dynamic SQL and XQuery statements. Several tools or methods give you the flexibility you need to capture, display, and analyze explain information.

Detailed optimizer information that allows for in-depth analysis of an access plan is stored in explain tables separate from the actual access plan itself. Use one or more of the following methods of getting information from the explain tables:

- Use Visual Explain to view explain snapshot information

  Invoke Visual Explain from the Control Center to see a graphical display of a query access plan. You can analyze both static and dynamic SQL and XQuery statements.

  Visual Explain allows you to view snapshots captured or taken on another platform. For example, a Windows client can graph snapshots generated on a DB2 for HP-UX server.
- Use the *db2exfmt* tool to display explain information in preformatted output.
- Use the *db2expln* and *dynexpln* tools

  To see the access plan information available for one or more packages of static SQL or XQuery statements, use the *db2expln* tool from the command line. *db2expln* shows the actual implementation of the chosen access plan. It does not show optimizer information.

  The *dynexpln* tool, which uses *db2expln* within it, provides a quick way to explain dynamic SQL or XQuery statements that contain no parameter markers. This use of *db2expln* from within *dynexpln* is done by transforming the input SQL or XQuery statement into a static statement within a pseudo-package. When this occurs, the information may not always be completely accurate. If complete accuracy is desired, use the explain facility.

  The *db2expln* tool does provide a relatively compact and English-like overview of what operations will occur at run-time by examining the actual access plan generated.
- Write your own queries against the explain tables

Writing your own queries allows for easy manipulation of the output and for comparison among different queries or for comparisons of the same query over time.

Note: The location of the command-line explain tools and others, such as *db2batch*, *dynexpln*, and *db2_all*, is in the *misc* subdirectory of the *sqllib* directory. If the tools are moved from this path, the command-line methods might not work.

The following table summarizes the different tools available with the DB2 explain facility and their individual characteristics. Use this table to select the tool most suitable for your environment and needs.

Table 60. Explain Facility Tools

Desired Characteristics	Visual Explain	Explain tables	db2exfmt	db2expln	dynexpln
GUI-interface	Yes				
Text output			Yes	Yes	Yes
"Quick and dirty" static SQL and XQuery analysis				Yes	
Static SQL and XQuery supported	Yes	Yes	Yes	Yes	
Dynamic SQL and XQuery supported	Yes	Yes	Yes	Yes	Yes*
CLI applications supported	Yes	Yes	Yes		
Available to DRDA® Application Requesters		Yes			
Detailed optimizer information	Yes	Yes	Yes		
Suited for analysis of multiple statements		Yes	Yes	Yes	Yes
Information accessible from within an application		Yes			
Note: * Indirectly using db2expln; there are some limitations.					

Related concepts:

- "dynexpln" on page 222
- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Related reference:

- Appendix F, "db2exfmt - Explain table format," on page 669
- "db2expln - SQL and XQuery Explain" on page 628

Guidelines for using explain information

You use explain information for the following two major purposes:

- To understand why application performance has changed
- To evaluate performance tuning efforts

Analysis of performance changes

To help you understand the reasons for changes in query performance, you need the before and after explain information which you can obtain by performing the following steps:

- Capture explain information for the query before you make any changes and save the resulting explain tables, or you might save the output from the `db2exfmt` explain tool.
- Save or print the current catalog statistics if you do not want to, or cannot, access Visual Explain to view this information. You might also use the `db2look` productivity tool to help perform this task.
- Save or print the data definition language (DDL) statements, including those for CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE TABLESPACE.

The information that you collect in this way provides a reference point for future analysis. For dynamic SQL or XQuery statements, you can collect this information when you run your application for the first time. For static SQL and XQuery statements, you can also collect this information at bind time. To analyze a performance change, you compare the information that you collected with information that you collect about the query and environment when you start your analysis.

As a simple example, your analysis might show that an index is no longer being used as part of the access path. Using the catalog statistics information in Visual Explain, you might notice that the number of index levels (NLEVELS column) is now substantially higher than when the query was first bound to the database. You might then choose to perform one of these actions:

- Reorganize the index
- Collect new statistics for your table and indexes
- Gather explain information when rebinding your query.

After you perform one of the actions, examine the access plan again. If the index is used again, performance of the query might no longer be a problem. If the index is still not used or if performance is still a problem, perform a second action and examine the results. Repeat these steps until the problem is resolved.

Evaluation of performance tuning efforts

You can take a number of actions to help improve query performance, such as adjusting configuration parameters, adding containers, collecting fresh catalog statistics, and so on.

After you make a change in any of these areas, you can use the explain facility to determine the impact, if any, that the change has on the access plan chosen. For example, if you add an index or materialized query table (MQT) based on the index guidelines, the explain data can help you determine whether the index or materialized query table is actually used as you expected.

Although the explain output provides information that allows you to determine the access plan that was chosen and its relative cost, the only way to accurately measure the performance improvement for a query is to use benchmark testing techniques.

Related concepts:

- "Materialized query tables" on page 11
- "Guidelines for capturing explain information" on page 231

- "Explain facility" on page 221
- "The Design Advisor" on page 43
- "The explain tables and organization of explain information" on page 225

Related reference:

- Appendix F, "db2exfmt - Explain table format," on page 669

The explain tables and organization of explain information

All explain information is organized around the concept of an explain instance. An explain instance represents one invocation of the explain facility for one or more SQL or XQuery statements. The explain information captured in one explain instance includes the compilation environment as well as the access plan chosen to satisfy the SQL or XQuery statement being compiled. For example, an explain instance might consist of any one of the following:

- All eligible SQL or XQuery statements in one package for static query statements. For SQL statements (including those that query XML data), you can capture explain information for SELECT, SELECT INTO, UPDATE, INSERT, VALUES, VALUES INTO, and DELETE statements. For XQuery statements, you can obtain explain information for XQUERY db2-fn:xmlcolumn and XQUERY db2-fn:sqlquery statements.
- One particular SQL statement for incremental bind SQL statements
- One particular SQL statement for dynamic SQL statements
- Each EXPLAIN SQL statement (whether dynamic or static)

Explain table information reflects the relationships between operators and data objects in the access plan. The following diagram shows the relationships between these tables.

Explain information is stored in the following tables:

Table 61. Relational tables that store explain data

Table Name	Description
EXPLAIN_ARGUMENT	Contains information about the unique characteristics for each individual operator, if any.
EXPLAIN_INSTANCE	The main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. Basic information about the source of the SQL or XQuery statements being explained and environment information is kept in this table.
EXPLAIN_OBJECT	Identifies those data objects required by the access plan generated to satisfy the SQL or XQuery statement.
EXPLAIN_OPERATOR	Contains all the operators needed to satisfy the SQL or XQuery statement by the query compiler.
EXPLAIN_PREDICATE	Identifies the predicates that are applied by a specific operator.

Table 61. Relational tables that store explain data (continued)

Table Name	Description
EXPLAIN_STATEMENT	Contains the text of the SQL or XQuery statement as it exists for the different levels of explain information. The original SQL or XQuery statement as entered by the user is stored in this table with the version used by the optimizer to choose an access plan. When an explain snapshot is requested, additional explain information is recorded to describe the access plan selected by the query optimizer. This information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table in the format required by Visual Explain. This format is not usable by other applications.
EXPLAIN_STREAM	Represents the input and output data streams between individual operators and data objects. The data objects themselves are represented in the EXPLAIN_OBJECT table. The operators involved in a data stream are represented in the EXPLAIN_OPERATOR table.
EXPLAIN_DIAGNOSTIC	Contains an entry for each diagnostic message produced for a particular instance of an explained statement in the EXPLAIN_STATEMENT table.
EXPLAIN_DIAGNOSTIC_DATA	Contains message tokens for specific diagnostic messages that are recorded in the EXPLAIN_DIAGNOSTIC table. The message tokens provide additional information that is specific to the execution of the SQL statement that generated the message.
ADVISE_WORKLOAD	Allows users to describe a workload to the database. Each row in the table represents an SQL or XQuery statement in the workload and is described by an associated frequency. The `db2advis` tool uses this table to collect and store workload information.
ADVISE_INSTANCE	Contains information about `db2advis` execution, including start time. Contains one row for each execution of `db2advis`.
ADVISE_INDEX	Stores information about recommended indexes. The table can be populated by the query compiler, the `db2advis` utility or a user. This table is used in two ways: • To get recommended indexes. • To evaluate indexes based on input about proposed indexes.
ADVISE_MQT	Contains the CREATE DDL, the query defining each recommended MQT, the statistics for each MQT such as COLSTATS (per column) in XML form, NUMROWS, and so on,, as well as the sampling query to obtain sampled statistics for each MQT.
ADVISE_TABLE	Stores the DDL for table creation using the final Design Advisor recommendations for recommended MQTs, MDCs and database partitions, depending on the options specified and the recommendations generated.
ADVISE_PARTITION	Stores virtual database partitions generated and evaluated by `db2advis`.

Note: Not all of the tables above are created by default. To create them, run the EXPLAIN.DDL script found in the *misc* subdirectory of the *sqllib* subdirectory.

Explain tables might be common to more than one user. However, the explain tables can be defined for one user, and then aliases can be defined for each additional user using the same name to point to the defined tables. Alternatively, the explain tables can be defined under the SYSTOOLS schema. The Explain facility will default to the SYSTOOLS schema if no other explain tables or aliases are found under the user's session ID for dynamic SQL or XQuery statements, or the statement authorization ID for static SQL or XQuery statements. Each user

sharing the common explain tables must have insert permission on those tables. Read permission for the common explain tables should also be limited, typically to users who analyze the explain information.

Related concepts:

- "Explain information for data objects" on page 227
- "Explain information for data operators" on page 230
- "Explain information for instances" on page 227
- "Explain facility" on page 221
- "SQL and XQuery Explain tools" on page 627

Explain information for data objects

A single access plan may use one or more data objects to satisfy the SQL or XQuery statement.

Object Statistics: The explain facility records information about the object, such as the following:

- The creation time
- The last time that statistics were collected for the object
- An indication of whether or not the data in the object is ordered (only table or index objects)
- The number of columns in the object (only table or index objects)
- The estimated number of rows in the object (only table or index objects)
- The number of pages that the object occupies in the buffer pool
- The total estimated overhead, in milliseconds, for a single random I/O to the specified table space where this object is stored
- The estimated transfer rate, in milliseconds, to read a 4K page from the specified table space
- Prefetch and extent sizes, in 4K pages
- The degree of data clustering with the index
- The number of leaf pages used by the index for this object and the number of levels in the tree
- The number of distinct full key values in the index for this object
- The total number of overflow records in the table

Related concepts:

- "Explain information for data operators" on page 230
- "Explain information for instances" on page 227
- "Guidelines for analyzing explain information" on page 233
- "The explain tables and organization of explain information" on page 225

Explain information for instances

Explain instance information is stored in the EXPLAIN_INSTANCE table. Additional specific information about each query statement in an instance is stored in the EXPLAIN_STATEMENT table.

Explain Instance Identification: The information provided by the following items helps you to uniquely identify each explain instance and to correlate the information for the query statements with a given invocation of the facility:

- The user who requested the explain information
- When the explain request began
- The name of the package that contains the explained query statement
- The SQL schema of the package that contains the explained query statement
- The version of the package that contains the statement
- Whether snapshot information was collected

Environmental Settings: Information about the database manager environment in which the query compiler optimized your queries is captured. The environmental information includes the following:

- The version and release number for the level of DB2
- The degree of parallelism for which the query was compiled

  The CURRENT DEGREE special register, the DEGREE bind option, the SET RUNTIME DEGREE API, and the *dft_degree* configuration parameter determine the degree of parallelism for which a particular query is compiled.
- Whether the query statement is dynamic or static
- The query optimization class used to compile the query
- The type of row blocking for cursors specified when compiling the query
- The isolation level in which the query runs
- The values of various configuration parameters when the query was compiled. The following parameters are recorded when an explain snapshot is taken:
  - Sort Heap Size (*sortheap*)
  - Average Number of Active Applications (*avg_appls*)
  - Database Heap (*dbheap*)
  - Maximum Storage for Lock List (*locklist*)
  - Maximum Percent of Lock List Before Escalation (*maxlocks*)
  - CPU Speed (*cpuspeed*)
  - Communications Bandwidth (*comm_bandwidth*)

Statement Identification: More than one query statement might have been explained for each explain instance. In addition to information that uniquely identifies the explain instance, the following information helps identify individual query statements:

- The type of statement: SELECT, DELETE, INSERT, UPDATE, positioned DELETE, positioned UPDATE
- The statement and section number of the package issuing the query statement, as recorded in SYSCAT.STATEMENTS catalog view

The QUERYTAG and QUERYNO fields in the EXPLAIN_STATEMENT table contain identifiers that are set as part of the explain process. For dynamic explain query statements submitted during a CLP or CLI session, when EXPLAIN MODE or EXPLAIN SNAPSHOT is active, the QUERYTAG is set to “CLP” or “CLI”. In this case, the QUERYNO value defaults to a number that is incremented by one or more for each statement. For all other dynamic explain query statements, which are not from CLP, CLI, or do not use the EXPLAIN query statement, QUERYTAG is set to blanks and QUERYNO is always “1”.

Cost Estimation: For each explained statement, the optimizer records an estimate of the relative cost of executing the chosen access plan. This cost is stated in an invented relative unit of measure called a *timeron*. No estimate of elapsed times is provided, for the following reasons:

- The query optimizer does not estimate elapsed time but only resource consumption.
- The optimizer does not model all factors that can affect elapsed time. It ignores factors that do not affect the efficiency of the access plan. A number of runtime factors affect the elapsed time, including the system workload, the amount of resource contention, the amount of parallel processing and I/O, the cost of returning rows to the user, and the communication time between the client and server.

Statement Text: Two versions of the text of the query statement are recorded for each statement explained. One version is the code that the query compiler receives from the application. The other version is reverse-translated from the internal compiler representation of the query. Although this translation looks similar to other query statements, it does not necessarily follow correct query language syntax nor does it necessarily reflect the actual content of the internal representation as a whole. This translation is provided only to allow you to understand the context in which the SQL and XQuery optimizer chose the access plan. To understand how the SQL and XQuery compiler has rewritten your query for better optimization, compare the user-written statement text to the internal representation of the query statement. The rewritten statement also shows you other elements in the environment affecting your statement, such as triggers and constraints. Some keywords used by this "optimized" text are:

$C*n*	The name of a derived column, where *n* represents an integer value.
$CONSTRAINT$	The tag used to indicate the name of a constraint added to the original query statement during compilation. Seen in conjunction with the $WITH_CONTEXT$ prefix.
$DERIVED.T*n*	The name of a derived table, where *n* represents an integer value.
$INTERNAL_FUNC$	The tag indicates the presence of a function used by the SQL and XQuery compiler for the explained query but not available for general use.
$INTERNAL_PRED$	The tag indicates the presence of a predicate added by the SQL and XQuery compiler during compilation of the explained query but not available for general use. An internal predicate is used by the compiler to satisfy additional context added to the original query statement because of triggers and constraints.
$INTERNAL_XPATH$	Shows an internal table function which takes a single input annotated XPath pattern as a parameter and returns a table with one or more columns that match the pattern.
RID	The tag used to identify the row identifier (RID) column for a particular row.
$TRIGGER$	The tag used to indicate the name of a trigger

added to the original query statement during compilation. Seen in conjunction with the $WITH_CONTEXT$ prefix.

$WITH_CONTEXT$(...) This prefix appears at the start of the text when additional triggers or constraints have been added into the original query statement. A list of the names of any triggers or constraints affecting the compilation and resolution of the query statement appears after this prefix.

Related concepts:

- "Explain information for data objects" on page 227
- "Explain information for data operators" on page 230
- "Guidelines for analyzing explain information" on page 233
- "The explain tables and organization of explain information" on page 225

Related reference:

- "avg_appls - Average number of active applications " on page 424
- "comm_bandwidth - Communications bandwidth " on page 498
- "cpuspeed - CPU speed " on page 499
- "dbheap - Database heap " on page 382
- "dft_degree - Default degree " on page 475
- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "locklist - Maximum storage for lock list " on page 383
- "sortheap - Sort heap size " on page 400

Explain information for data operators

A single access plan may perform several operations on the data to satisfy the SQL or XQuery statement and provide results back to you. The query compiler determines the operations required, such as a table scan, an index scan, a nested loop join, or a group-by operator.

In addition to showing the operators used in an access plan and information about each operator, explain information also shows the cumulative effects of the access plan.

Estimated Cost Information: The following estimated cumulative costs can be displayed for the operators. These costs are for the chosen access plan, up to and including the operator for which the information is captured.

- The total cost (in timerons)
- The number of page I/Os
- The number of CPU instructions
- The cost (in timerons) of fetching the first row, including any initial overhead required
- The communication cost (in frames).

Timerons are an invented relative unit of measure. Timerons are determined by the optimizer based on internal values such as statistics that change as the database is used. As a result, the timerons measure for a SQL or XQuery statement are not guaranteed to be the same every time the estimated cost in timerons is determined.

Operator Properties: The following information is recorded by the explain facility to describe the properties of each operator:

- The set of tables that have been accessed
- The set of columns that have been accessed
- The columns on which the data is ordered, if the optimizer determined that this ordering can be used by subsequent operators
- The set of predicates that have been applied
- The estimated number of rows that will be returned (cardinality)

Related concepts:

- "Explain information for data objects" on page 227
- "Explain information for instances" on page 227
- "Guidelines for analyzing explain information" on page 233
- "The explain tables and organization of explain information" on page 225

Guidelines for capturing explain information

Explain data is captured if you request it when an SQL or XQuery statement is compiled. Consider how you expect to use the captured information when you request explain data.

Notes:

1. If incremental bind SQL or XQuery statements are compiled at run time, data is placed in the explain tables at run time and not bind time. For these statements, the explain table qualifier and authorization ID inserted is that of the package owner and not that of the user running the package.
2. Explain information is captured only when the SQL or XQuery statement is compiled. After the initial compilation, dynamic query statements are recompiled when a change to the environment requires it, or when the Explain facility is active. If you issue the same PREPARE statement for the same query statement, the statement is compiled and explain data is captured every time this statement is prepared or executed.
3. If a package is bound using the bind option REOPT ONCE/ALWAYS, SQL or XQuery statements containing host variables, parameter markers, or special registers will be compiled and the access path will be created using real values of these variables if they are known, and using default estimates if the values are not known at compilation time.
4. If the FOR REOPT ONCE clause is used, then an attempt is made to match the specified SQL or XQuery statement against the same statement in the package cache. The values of this already reoptimized cached query statement will be used to reoptimize the specified query statement. The Explain tables will contain the newly generated reoptimized access plan and the values used for this reoptimization, if the user has the required access privileges.
5. In a multi-partition system, the statement should be explained on the same database partition on which it was originally compiled and reoptimized using REOPT ONCE, otherwise an error will be returned.

Capturing information in the explain tables

- **Static or incremental bind SQL and XQuery statements**:

  Specify either EXPLAIN ALL or EXPLAIN YES options on the BIND or the PREP commands or include a static EXPLAIN statement in the source program.
- **Dynamic SQL and XQuery statements**:

Explain table information is captured in any of the following cases:

– The CURRENT EXPLAIN MODE special register is set to:
  - YES: The SQL and XQuery compiler captures explain data and executes the query statement.
  - EXPLAIN: The SQL and XQuery compiler captures explain data, but does not execute the query statement.
  - RECOMMEND INDEXES: The SQL and XQuery compiler captures explain data and the recommended indexes are placed in the ADVISE_INDEX table, but the query statement is not executed.
  - EVALUATE INDEXES: The SQL and XQuery compiler uses indexes placed by the user in the ADVISE_INDEX table for evaluation. In EVALUATE INDEXES mode, all dynamic statements are explained as if these virtual indexes were available. The query compiler then chooses to use the virtual indexes if the improve the performance of the statements. Otherwise, the indexes are ignored. To find out if proposed indexes are useful, review the EXPLAIN results.
  - REOPT: The query compiler captures explain data for static or dynamic SQL or XQuery statements during statement reoptimization at execution time, when actual values for the host variables, special registers, or parameter markers are available.

• The EXPLAIN ALL option has been specified on the BIND or PREP command. The query compiler captures explain data for dynamic SQL and XQuery at run-time, even if the CURRENT EXPLAIN MODE special register is set to NO. The SQL or XQuery statement also executes and returns the results of the query.

Capturing explain snapshot information

When an explain snapshot is requested, explain information is stored in the SNAPSHOT column of the EXPLAIN_STATEMENT table in the format required by Visual Explain. This format is not usable by other applications. Additional information on the contents of the explain snapshot information is available from Visual Explain itself. This information includes information about data objects and data operators.

Explain snapshot data is captured when an SQL or XQuery statement is compiled and explain data has been requested, as follows:

• **Static or incremental bind SQL and XQuery statements**:

  An explain snapshot is captured when either EXPLSNAP ALL or EXPLSNAP YES clauses are specified on the BIND or the PREP commands or when the source program includes a static EXPLAIN statement that uses a FOR SNAPSHOT or a WITH SNAPSHOT clause.

• **Dynamic SQL and XQuery statements**:

  An explain snapshot is captured in any of the following cases:

  – You issue an EXPLAIN statement with a FOR SNAPSHOT or a WITH SNAPSHOT clause. With the FOR SNAPSHOT clause, only explain snapshot information is captured. With the WITH SNAPSHOT clause, all explain information is captured in addition snapshot information.
  – The CURRENT EXPLAIN SNAPSHOT special register is set to:
    - YES: The query compiler captures snapshot explain data and executes the SQL or XQuery statement.
    - EXPLAIN: The query compiler captures snapshot explain data, but does not execute the SQL or XQuery statement.

– You specify the EXPLSNAP ALL option on the BIND or PREP command. The query compiler captures snapshot explain data at run-time, even if the setting of the CURRENT EXPLAIN SNAPSHOT special register is NO. It also executes the SQL or XQuery statement.

Related concepts:

- "The Design Advisor" on page 43
- "Guidelines for analyzing explain information" on page 233
- "Guidelines for using explain information" on page 223
- "Explain facility" on page 221
- "SQL and XQuery Explain tools" on page 627
- "The explain tables and organization of explain information" on page 225

Guidelines for analyzing explain information

The primary use of explain information is analysis of the access paths for query statements. There are a number of ways in which analyzing the explain data can help you to tune your queries and environment. Consider the following kind of analysis:

- **Index use**

  The proper indexes can significantly benefit performance. Using the explain output, you can determine if the indexes you have created to help a specific set of queries are being used. In the explain output, you should look for index usage in the following areas:

  – Join predicates
  – Local predicates
  – GROUP BY clause
  – ORDER BY clause
  – WHERE XMLEXISTS clause
  – The select list.

  You can also use the explain facility to evaluate whether a different index might be used instead of an existing index or no index at all. After you create a new index, use the RUNSTATS command to collect statistics for that index and recompile the query. Over time you may notice through the explain data that instead of an index scan, a table scan is now being used. This can result from a change in the clustering of the table data. If the index that was previously being used now has a low cluster ratio, you may want to reorganize the table to cluster its data according to that index, use the RUNSTATS command to collect statistics for both index and table, and then recompile the query. To determine whether reorganizing table has improved the access plan, re-examine the explain output for the recompiled query.

- **Access type**

  Analyze the explain output and look for types of access to the data that are not usually optimal for the type of application you are running. For example:

  – **Online transaction processing (OLTP) queries**

    OLTP applications are prime candidates to use index scans with range delimiting predicates, because they tend to return only a few rows that are qualified using an equality predicate against a key column. If your OLTP queries are using a table scan, you may want to analyze the explain data to determine the reasons why an index scan was not used.

- **Browse-only queries**

  The search criteria for a "browse" type query may be very vague, causing a large number of rows to qualify. If users usually look at only a few screens of the output data, you might specify that the entire answer set need not be computed before some results are returned. In this case, the goals of the user are different from the basic operating principle of the optimizer, which attempts to minimize resource consumption for the entire query, not just the first few screens of data.

  For example, if the explain output shows that both merge scan join and sort operators were used in the access plan, then the entire answer set will be materialized in a temporary table before any rows are returned to the application. In this case, you can attempt to change the access plan by using the OPTIMIZE FOR clause on the SELECT statement. If you specify this option, the optimizer can attempt to choose an access plan that does not produce the entire answer set in a temporary table before returning the first rows to the application.

- **Join methods**

  If a query joins two tables, check the type of join being used. Joins that involve more rows, such as those in decision-support queries, usually run faster with a hash join or a merge join. Joins that involve only a few rows, such as OLTP queries, typically run faster with nested-loop joins. However, there may be extenuating circumstances in either case, such as the use of local predicates or indexes, that might change how these typical joins work.

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Explain facility" on page 221
- "SQL and XQuery Explain tools" on page 627

Usage notes for dynexpln

To explain dynamic statements, `dynexpln` creates a static application for the statements and then invokes `db2expln`. To create the static statements, `dynexpln` generates a trivial C program with the statements and then calls the DB2 precompiler to create the package. (The generated C program is not complete and cannot be compiled; it only contains enough information for the precompiler to build the package.)

The following are common messages displayed by `dynexpln`:

- All error messages from `db2expln`.

  Since `dynexpln` invokes `db2expln`, it is possible to see most of `db2expln`'s error messages.

- `Error connecting to the database.`

  This message will appear in the output if an error occurred connecting to the database. A CLI error message will also be displayed indicating why the connection could not be completed. Correct the cause of the error and run `dynexpln` again.

- `The file "<filename>" must be removed before dynexpln will run.`

  This message will appear if the given file exists at the time `dynexpln` is run. Remove the file or change the value of the `DYNEXPLN_PACKAGE` environment variable to change the name of the file which will be created and run `dynexpln` again.

- `The package "<creator>.<package>" must be dropped before dynexpln will run.`

  This message will appear if the given package exists at the time `dynexpln` is run. Drop the package and run or change the value of the **DYNEXPLN_PACKAGE** environment variable to change the name of the package which will be created and run `dynexpln` again.
- `Error writing file "<filename>".`

  This message will appear if the given file cannot be written to. Ensure that `dynexpln` can write files in the current directory and run it again.
- `Error reading input file "<filename>".`

  This message will appear if the file given with the **-f** option cannot be read from. Ensure that the file exists and that `dynexpln` can read it. Then run `dynexpln` again.

Environment Variables: There are two different environment variables that can be used in conjunction with `dynexpln`:

- **DYNEXPLN_OPTIONS** are the SQL and XQuery precompiler options you use when building the package for your statements. Use the same syntax variable as you would when issuing a PREP command through CLP.

  For example: `DYNEXPLN_OPTIONS="OPTLEVEL 5 BLOCKING ALL"`
- **DYNEXPLN_PACKAGE** is the name of the package which is created in the database. The statements to be described are placed in this package. If this variable is not defined, the package is given a default value of **DYNEXPLN**. (Only the first eight characters of the name in this environment variable are used.)

  The name is also used to create the names for the intermediate files that `dynexpln` uses.

Related concepts:

- "dynexpln" on page 222
- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Chapter 11. Configuring DB2 instances and databases

Configuration parameters

When a DB2 database instance or a database is created, a corresponding configuration file is created with default parameter values. You can modify these parameter values to improve performance and other characteristics of the instance or database.

Configuration files contain parameters that define values such as the resources allocated to the DB2 database products and to individual databases, and the diagnostic level. There are two types of configuration files:

- The database manager configuration file for each DB2 instance
- The database configuration file for each individual database.

The *database manager configuration file* is created when a DB2 instance is created. The parameters it contains affect system resources at the instance level, independent of any one database that is part of that instance. Values for many of these parameters can be changed from the system default values to improve performance or increase capacity, depending on your system's configuration.

There is one database manager configuration file for each client installation as well. This file contains information about the client enabler for a specific workstation. A subset of the parameters available for a server are applicable to the client.

Database manager configuration parameters are stored in a file named `db2systm`. This file is created when the instance of the database manager is created. In UNIX-based environments, this file can be found in the `sqllib` subdirectory for the instance of the database manager. In Windows, the default location of this file is the instance subdirectory of the `sqllib` directory. If the DB2INSTPROF variable is set, the file is in the `instance` subdirectory of the directory specified by the DB2INSTPROF variable.

In a partitioned database environment, this file resides on a shared file system so that all database partition servers have access to the same file. The configuration of the database manager is the same on all database partition servers.

Most of the parameters either affect the amount of system resources that will be allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

A *database configuration file* is created when a database is created, and resides where that database resides. There is one configuration file per database. Its parameters specify, among other things, the amount of resource to be allocated to that database. Values for many of the parameters can be changed to improve performance or increase capacity. Different changes may be required, depending on the type of activity in a specific database.

Parameters for an individual database are stored in a configuration file named SQLDBCON. This file is stored along with other control files for the database in the SQLnnnnn directory, where nnnnn is a number assigned when the database was created. Each database has its own configuration file, and most of the parameters in the file specify the amount of resources allocated to that database. The file also contains descriptive information, as well as flags that indicate the status of the database.

In a partitioned database environment, a separate SQLDBCON file exists for each database partition. The values in the SQLDBCON file may be the same or different at each database partition, but the recommendation is that the database configuration parameter values be the same on all database partitions.

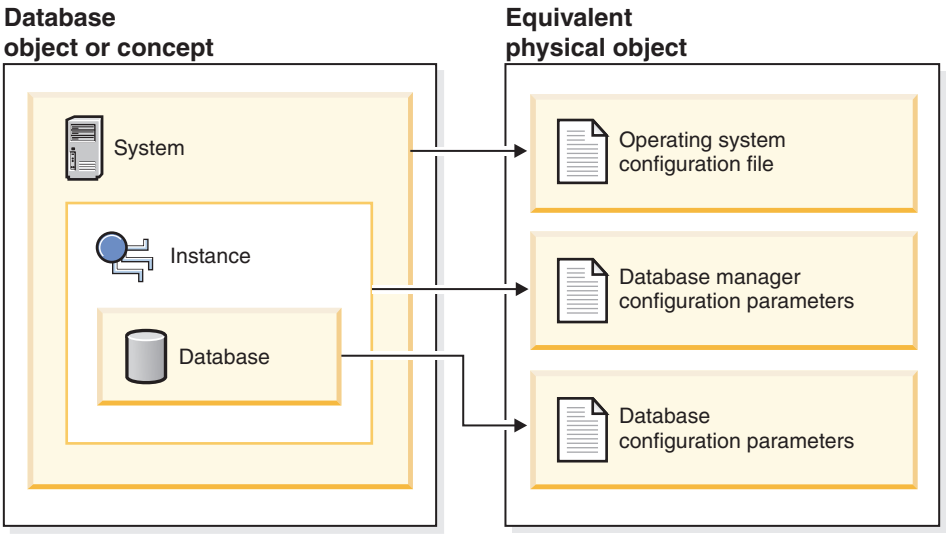


Figure 26. Relationship between database objects and configuration files

Related concepts:

- "Configuration parameters that affect query optimization" on page 158

Related tasks:

- "Configuring DB2 with configuration parameters" on page 238

Configuring DB2 with configuration parameters

The disk space and memory allocated by the database manager on the basis of default values of the parameters may be sufficient to meet your needs. In some situations, however, you may not be able to achieve maximum performance using these default values.

Since the default values are oriented towards machines that have relatively small memory resources and are dedicated as database servers, you may need to modify these values if your environment has:

- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

A good starting point for tuning your configuration is to use the Configuration Advisor or the AUTOCONFIGURE command which will generate values for parameters based on your responses to questions about workload characteristics.

Some configuration parameters can be set to *automatic*. DB2 will then automatically adjust these parameters to reflect the current resource requirements.

Database manager configuration parameters are stored in a file named `db2systm`. Database configuration parameters are stored in a file named `SQLDBCON`. These files cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.

Attention: If you edit `db2systm` or `SQLDBCON` using a method other than those provided by DB2, you may make the database unusable. **We strongly recommend** that you do not change these files using methods other than those documented and supported by DB2.

You may use one of the following methods to reset, update, and view configuration parameters:

- Using the Control Center. The Configure Instance notebook can be used to set the database manager configuration parameters on either a client or a server. The Configure Database notebook can be used to alter the value of database configuration parameters. The DB2 Control Center also provides the Configuration Advisor to alter the value of configuration parameters. This advisor generates values for parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database.

  In a partitioned database environment, the `SQLDBCON` file exists for each database partition. The Configure Database notebook will change the value on all database partitions if you launch the notebook from the database object in the tree view of the Control Center. If you launch the notebook from a database partition object, then it will only change the values for that database partition. (We recommend, however, that the configuration parameter values be the same on all database partitions.)

  Note: The Configuration Advisor is not available in the partitioned database environment.
- Using the command line processor. Commands to change the settings can be quickly and conveniently entered:

  For database manager configuration parameters:

  – GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)
  – UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
  – RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG) to reset *all* database manager parameters to their default values
  – AUTOCONFIGURE.

  For database configuration parameters:

  – GET DATABASE CONFIGURATION (or GET DB CFG)

  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG) to reset *all* database parameters to their default values
  - AUTOCONFIGURE.
- Using application programming interfaces (APIs). The APIs can be called from an application or a host-language program. Call the following DB2 APIs to view or update configuration parameters:
  - db2AutoConfig - Access the Configuration Advisor
  - db2CfgGet - Get the database manager or database configuration parameters
  - db2CfgSet - Set the database manager or database configuration parameters
- Using the Configuration Assistant (for database manager configuration parameters). You can only use the Configuration Assistant to set the database manager configuration parameters on a client.

For some database manager configuration parameters, the database manager must be stopped (`db2stop`) and then restarted (`db2start`) for the new parameter values to take effect.

For some database parameters, changes will only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes will take effect.

Other parameters can be changed online; these are called *configurable online configuration parameters*.

If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command will be to apply the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

To change a database manager configuration parameter online:

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

For clients, changes to the database manager configuration parameters take effect the next time the client connects to a server.

If you change a configurable online database configuration parameter while connected, the default behavior is to apply the change online, wherever possible. You should note that some parameter changes may take a noticeable amount of time to take effect due to the overhead associated with allocating space. To change configuration parameters online from the command line processor, a connection to the database is required. To change a database configuration parameter online:

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

Each configurable online configuration parameter has a *propagation class* associated with it. The propagation class indicates when you can expect a change to the configuration parameter to take effect. There are three propagation classes:

- **Immediate**: Parameters that change immediately upon command or API invocation. For example, *diaglevel* has a propagation class of immediate.
- **Statement boundary**: Parameters that change on statement and statement-like boundaries. For example, if you change the value of *sortheap*, all new requests will start using the new value.
- **Transaction boundary**: Parameters that change on transaction boundaries. For example, a new value for *dl_expint* is updated after a COMMIT statement.

Changing some database configuration parameters can influence the access plan chosen by the SQL and XQuery optimizer. After changing any of these parameters, you should consider rebinding your applications to ensure the best access plan is being used for your SQL and XQuery statements. Any parameters that were modified online (for example, by using the UPDATE DATABASE CONFIGURATION IMMEDIATE command) will cause the SQL and XQuery optimizer to choose new access plans for new query statements. However, the query statement cache will not be purged of existing entries. To clear the contents of the query cache, use the FLUSH PACKAGE CACHE statement.

While new parameter values may not be immediately effective, viewing the parameter settings (using GET DATABASE MANAGER CONFIGURATION or GET DATABASE CONFIGURATION commands) will always show the latest updates. Viewing the parameter settings using the SHOW DETAIL clause on these commands will show both the latest updates and the values in memory.

Note: A number of configuration parameters (for example, *userexit*) are described as having acceptable values of either "Yes" or "No", or "On" or "Off" in the help and other DB2 documentation. To clarify, "Yes" should be considered equivalent to "On" and "No" should be considered equivalent to "Off".

Related concepts:

- "Configuration parameters" on page 237

Related reference:

- "Configuration parameters summary" on page 264
- "FLUSH PACKAGE CACHE statement" in *SQL Reference, Volume 2*
- "AUTOCONFIGURE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "RESET DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "db2AutoConfig API - Access the Configuration Advisor" in *Administrative API Reference*
- "db2CfgGet API - Get the database manager or database configuration parameters" in *Administrative API Reference*
- "db2CfgSet API - Set the database manager or database configuration parameters" in *Administrative API Reference*
- "AUTOCONFIGURE command" in *Command Reference*

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Configuring parameters dynamically

DB2 allows you to take advantage of dynamic configuration. You can change certain configuration parameters in a DB2 database or instance while that database is running, accepting connections or processing transactions. Following is an example of how you might take advantage of dynamic configuration features in DB2.

In this scenario, a database server is configured with 4 GB of memory, of which 3.5 GB are available to the database manager. There are 8 processors. The database server is dedicated to a single database called DB2DYN running on instance DB2INST. The database workload varies throughout the day and week, as follows:

- The daytime workload (05:00-20:00) includes many connections and concurrent transactions.
- The end of day workload (20:00-24:00) includes summary reports and decision-support queries, with few connections and transactions.
- The daily maintenance workload (24:00-05:00) includes online load operations, incremental backups, index creations, and so on.
- The weekly maintenance workload includes large table reorganization operations, runstats operations, and larger index creations.

Given these workload characteristics, you could configure the system as shown in the following table.

	Daytime (05:00 - 20:00)	**End of Day (20:00 - 24:00)**	**Daily Maintenance (24:00 - 05:00)**	**Weekly Maintenance (Sundays)**
Database activity	Heavy transaction workload	Decision-support queries	Load, backup, index creation	Reorg and runstats executed in buffer pool 1
Buffer pool 1 (MB)	1000	500	500	2000
Buffer pool 2 (MB)	1000	500	500	200
Sort heap (MB)	0.1	20	200	200
Catalog cache (MB)	200	200	50	50
Package cache (MB)	800	200	200	200
Utility heap (MB)	0	0	1000	0
Diag level	1	3	4	4

You could begin by setting the database configuration parameter *database_memory* to 3.5 GB, thereby reserving this specified amount of available memory for the database. This is a one time operation which, when complete, will give the database 3.5 gigabytes (or 917 504 4-kilobyte pages) of reserved memory for buffer pool creation or configuration adjustment.

```
db2start
db2 update db cfg for db2dyn using database_memory 917504
db2stop
```

You could then use the following scripts to transition the database from one configuration to another. (These scripts can be scheduled to run at the appropriate times.)

MorningConfiguration.sh

```
# This script is used to prepare
# the database server for the
# morning configuration,
# for a workload consisting of
# a large number of OLTP connections and
# concurent transactions.

db2 connect to db2dyn

db2 update db cfg using sortheap 25
db2 update db cfg using util_heap_sz 32
db2 alter bufferpool bufferpool1 size 262144
db2 commit
db2 update db cfg using catcachesz 51200
db2 update db cfg using pkcachesz 204800
db2 alter bufferpool bufferpool2 size 262144
db2 commit
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 1
db2 get dbm cfg show detail
db2 detach
```

EveningConfiguration.sh

```
# This script is used to prepare
# the database server for the
# evening configuration,
# for a workload consisting of
# decision-support queries.

db2 connect to db2dyn

db2 alter bufferpool bufferpool1 size 131072
db2 commit
db2 alter bufferpool bufferpool2 size 131072
db2 commit
db2 update db cfg using pkcachesz 51200
db2 update db cfg using catcachesz 51200
db2 update db cfg using util_heap_sz 32
db2 update db cfg using sortheap 5120
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 3
db2 get dbm cfg show detail
db2 detach
```

NightTimeConfiguration.sh

```
# This script is used to prepare
# the database server for the
# daily maintenance configuration,
# for a workload consisting of
# index creation and load and backup
# operations.

db2 connect to db2dyn

db2 alter bufferpool bufferpool1 size 131072
db2 commit
db2 alter bufferpool bufferpool2 size 131072
db2 commit
db2 update db cfg using catcachesz 51200
db2 update db cfg using pkcachesz 51200
db2 update db cfg using sortheap 51200
db2 update db cfg using util_heap_sz 262144
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 4
db2 get dbm cfg show detail
db2 detach
```

MaintenanceConfiguration.sh

```
# This script is used to prepare
# the database server for the
# weekly maintenance configuration,
# for a workload consisting of
# reorg and runstats operations executed
# in buffer pool 1.

db2 connect to db2dyn

db2 update db cfg using util_heap_sz 32
db2 alter bufferpool bufferpool2 size 51200
db2 commit
db2 update db cfg using sortheap 5120
db2 update db cfg using catcachesz 51200
db2 update db cfg using pkcachesz 51200
db2 alter bufferpool bufferpool1 size 524288
db2 commit
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 4
db2 get dbm cfg show detail
db2 detach
```

Note that the order of operations differs from one script to the next. Operations that reduce memory requirements should be completed before operations that increase memory requirements; otherwise, the increases might fail.

All buffer pool resizings are followed by a commit operation, because buffer pool changes are SQL operations and are part of the transaction.

The package cache is flushed after each reconfiguration to signal the optimizer that the configuration has undergone a significant change, and that all existing dynamic SQL and XQuery access plans are now invalid.

Dynamic database manager reconfiguration operations are performed while an application is attached to the instance, and dynamic database operations are performed while an application is connected to the database.

The SHOW DETAIL clause on the GET DATABASE MANAGER CONFIGURATION command, or the GET DATABASE CONFIGURATION command, can be used to verify that a dynamic reconfiguration operation has taken effect.

Other database configuration parameters that could have been changed dynamically in this scenario include:

- The *locklist* parameter, which can be updated dynamically if the database experiences frequent lock escalations.
- The *dft_queryopt* parameter, which applies to new connections only, and can be used to increase the optimization level prior to the start of the decision-support workload.
- The *dft_degree* parameter, which also applies to new connections only, and can be used to provide decision-support queries with added intra-query parallelism.

Another option for addressing changing workload characteristics is to enable the self tuning memory feature, which distributes memory among different memory areas in response to changes in workload requirements.

Related concepts:

- "Self tuning memory" on page 255

Generating recommendations for database configuration

The Configuration Advisor is used to make recommendations for the initial values of the buffer pool size, database configuration parameters, and database manager configuration parameters. The recommended values can be displayed or applied by using the APPLY option. The recommendations are based on input that you provide and system information that the advisor gathers.

The values suggested by the Configuration Advisor are relevant for only one database per instance. If you want to use this advisor on more than one database, each database should belong to a separate instance.

In Version 9.1, the Configuration Advisor is automatically invoked when you create a database. To disable this feature for a given database, or to explicitly enable it, use the **db2set** command. Examples:

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

Prerequisites:

You can use the AUTOCONFIGURE option on an existing database or as an option to the **CREATE DATABASE** command.

To configure your database you must have either SYSADM, SYSCTRL, or SYSMAINT authority.

Procedure:

The Configuration Advisor can be run through the **AUTOCONFIGURE** command on the command line processor (CLP), through the Configuration Advisor GUI in the Control Center, or by calling the **db2AutoConfig** API.

To open the Configuration Advisor from the Control Center:

1. Expand the object tree until you find the database object for which you would like DB2 to provide configuration recommendations.
2. Right-click the database and select **Configure Advisor** from the pop-up menu. The Configuration Advisor opens.

Detailed information is provided through the online help facility within the Control Center.

To request configuration recommendations using the command line, enter:

```
AUTOCONFIGURE
   USING <input_keyword> <param_value>
   APPLY <value>
```

The following is an example of an **AUTOCONFIGURE** command that requests configuration recommendations based on input about how the database is used, but specifies that the recommendations should not be applied:

```
DB2 AUTOCONFIGURE USING
     MEM_PERCENT 60
     WORKLOAD_TYPE MIXED
     NUM_STMTS 500
     ADMIN_PRIORITY BOTH
     IS_POPULATED YES
     NUM_LOCAL_APPS 0
     NUM_REMOTE_APPS 20
     ISOLATION RR
     BP_RESIZEABLE YES
APPLY NONE
```

Related concepts:

- "Automatic features enabled by default" in *Administration Guide: Planning*

Related tasks:

- "Creating a database" in *Administration Guide: Implementation*

Related reference:

- "Configuration Advisor sample output" in *Administration Guide: Implementation*
- "AUTOCONFIGURE command" in *Command Reference*
- "db2AutoConfig API - Access the Configuration Advisor" in *Administrative API Reference*

Configuring DB2 memory allocation

Memory allocation in DB2

Memory allocation and de-allocation occurs at various times in DB2. Memory may be allocated to a particular memory area when a specified event occurs, such as when an application connects, or it may be re-allocated based on a change in a configuration parameter setting.

The figure below shows the different areas of memory that the database manager allocates for various uses and the configuration parameters that allow you to control the size of this memory. Note that in an Enterprise Server Edition environment that comprises multiple logical database partitions, each database partition has its own Database Manager Shared Memory set.

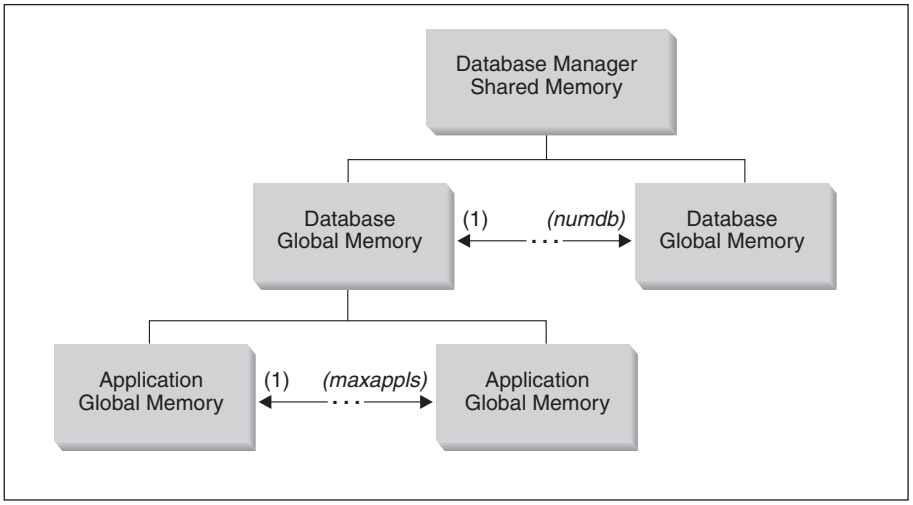


Figure 27. Types of memory used by the Database Manager

Memory is allocated for each instance of the database manager when the following events occur:

- **When the database manager is started (`db2start`):** Database manager global shared memory (also known as instance shared memory) is allocated and remains allocated until the database manager is stopped (`db2stop`). This area contains information that the database manager uses to manage activity across all database connections. Instance shared memory can be controlled by the *instance_memory* configuration parameter. By default, this parameter is set to `automatic` so that DB2 calculates the amount of memory allocated for the instance.
- **When a database is activated or connected to for the first time:** Database global memory is allocated. Database global memory is used across all applications that connect to the database. The size of the database global memory is specified by the *database_memory* configuration parameter. By default, this parameter is set to *automatic* so that DB2 calculates the amount of memory allocated for the database. You can set *database_memory* to allocate more memory than is needed initially so that the additional memory can be dynamically distributed later.

  Although the total amount of database global memory cannot be increased or decreased while the database is active, memory for areas contained in database

global memory can be adjusted. The following memory areas can be dynamically adjusted, for example, to decrease memory allocated to one area and increase memory in another area.

- Buffer pools (using the ALTER BUFFERPOOL DDL statement)
- Database heap (including log buffers)
- Utility heap
- Package cache
- Catalog cache
- Lock list (This memory area can only be increased dynamically, and not decreased.)

In an environment in which the database manager intra-partition parallelism configuration parameter (intra_parallel) is enabled, or in an environment in which the connection concentrator is enabled, the shared sort heap is also allocated as part of the database global memory.

- **When an application connects to a database:** In a partitioned database environment, in a non-partitioned database with the database manager intra-partition parallelism configuration parameter (*intra_parallel*) enabled, or in an environment in which the connection concentrator is enabled, multiple applications can be assigned to *application groups* to share memory. Each application group has its own allocation of shared memory. In the application-group shared memory, each application has its own application control heap but uses the shared heap of the application group which improves the efficiency of cache and memory usage.

  The following three database configuration parameters determine the size of the application group memory:

  - The *appgroup_mem_sz* parameter, which specifies the size of the shared memory for the application group
  - The *groupheap_ratio* parameter, which specifies the percent of the application-group shared memory allowed for the shared heap
  - The *app_ctl_heap_sz* parameter, which specifies the size of the control heap for each application in the group.

  The database manager configuration parameter *max_connections* sets an upper limit on the number of applications that can connect to a database. Since each application that attaches to a database involves the allocation of some memory, allowing a larger number of concurrent applications will potentially use more memory.

  To a certain extent, the maximum number of applications is also governed by the database manager configuration parameter *maxagents* or *max_coordagents* for partitioned database environments. The *maxagents* parameter sets an upper limit to the total number of database manager agents in a database partition. These database manager agents include active coordinator agents, subagents, inactive agents, and idle agents.

- **When an agent is created:** Agent private memory is allocated for an agent when the agent is assigned as the result of a connect request or a new SQL request in a parallel environment. Agent private memory is allocated for the agent and contains memory that is used only by this specific agent, such as the sort heap and the application heap.

  When a database is already in use by one application, only agent private memory and application global shared memory is allocated for subsequent connecting applications.

The figure also specifies the following configuration parameter settings, which limit the amount of memory that is allocated for each type of memory area. Note that in a partitioned database environment, this memory is allocated on each database partition.

- *instance_memory*

  This parameter specifies how much memory is allocated for instance management.
- *numdb*

  This parameter specifies the maximum number of concurrent active databases that different applications can use. Because each database has its own global memory area, the amount of memory that might be allocated increases if you increase the value of this parameter.
- *maxappls*

  This parameter specifies the maximum number of applications that can simultaneously connect to a single database. It affects the amount of memory that might be allocated for agent private memory and application global memory for that database. Note that this parameter can be set differently for every database.
- *maxagents* and *max_coordagents* for parallel processing

  These parameters limit the number of database manager agents that can exist simultaneously across all active databases in an instance. Together with *maxappls*, these parameters limit the amount of memory allocated for agent private memory and application global memory.

The memory tracker, invoked by the db2mtrk command, allows you to view the current allocation of memory within the instance, including the following types of information for each memory pool:

- Current size
- Maximum size (hard limit)
- Largest size (high water mark)

Related concepts:

- "Database manager shared memory" on page 249
- "The FCM buffer pool and memory requirements" on page 252
- "Tuning memory allocation parameters" on page 253

Related reference:

- "db2mtrk - Memory tracker command" in *Command Reference*

Database manager shared memory

Database manager shared memory is organized into several different memory areas. The following figure shows how database manager shared memory is allocated. The configuration parameters shown allow you to control the size of this memory by limiting the number and size of memory segments which are portions of logical memory.

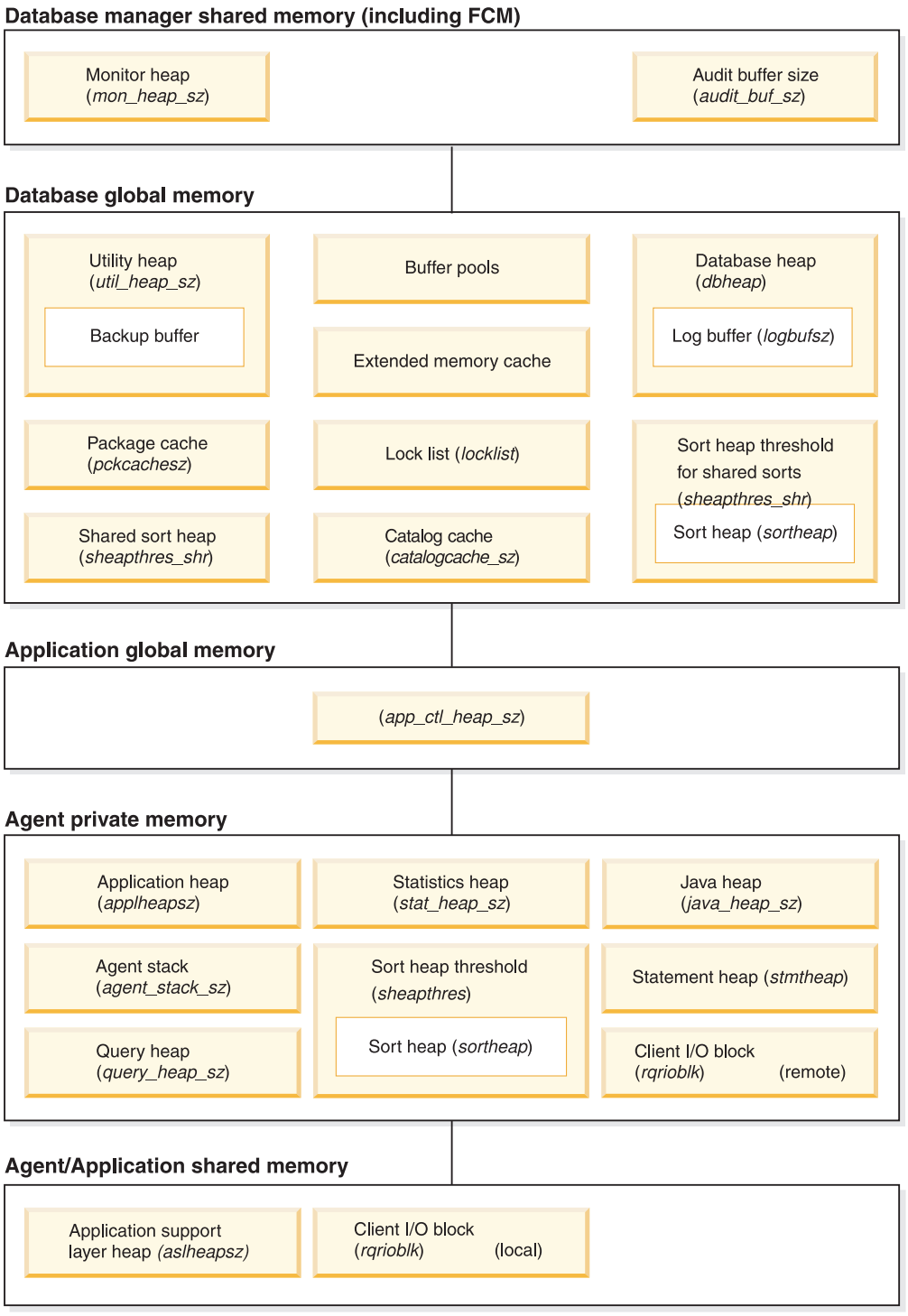


Figure 28. How memory is used by the database manager

Audit buffer

This memory area is used to size the buffer used in database auditing activities. The size of this buffer is determined by the *audit_buf_sz* configuration parameter.

Monitor heap

This memory area is used database system monitoring data. The size of this area is determined by the *mon_heap_sz* configuration parameter.

Fast communication manager (FCM) buffer pool

For partitioned database systems, the fast communications manager (FCM) requires substantial memory space, especially if the value of *fcm_num_buffers* is large. The FCM memory requirements are either allocated from the FCM Buffer Pool, or from both the Database Manager Shared Memory and the FCM Buffer Pool, depending on whether or not the partitioned database system uses multiple logical nodes. If there are multiple logical nodes, then a separate FCM Buffer Shared memory area is allocated for all logical nodes to share.

Database Global Memory

Database Global Memory is affected by the following configuration parameters:

- The *database_memory* parameter provides a lower bound for the size of the database global memory.
- The following parameters or factors specify the maximum size of memory segments:
  - The size of the buffer pools.
  - Maximum Storage for Lock List (*locklist*)
  - Database Heap (*dbheap*)
  - Utility Heap Size (*util_heap_sz*)
  - Package Cache Size (*pckcachesz*)
  - Shared Sort Heap (*sheapthres_shr*)
  - Catalog cache (*catalogcache_sz*)

Application Global Memory

Application Global Memory is affected by the Application Control Heap Size (*app_ctl_heap_sz*) configuration parameter.

For parallel systems, space is also required for the application control heap, which is shared between the agents that are working for the same application at one database partition. The heap is allocated when a connection is requested by the first agent to receive a request from the application. The agent can be either a coordinating agent or a subagent.

Agent Private Memory

- The number of memory segments is limited by the lower of:
  - The total of the *maxappls* configuration parameter for all active databases, that specifies the maximum number of active applications permitted.
  - The value of the *maxagents* configuration parameter, which specifies the maximum number of agents permitted.
- The maximum size of memory segments is determined by the values of the following parameters:
  - Application Heap Size (*applheapsz*)
  - Sort Heap Size (*sortheap*)
  - Statement Heap Size (*stmtheap*)
  - Statistics Heap Size (*stat_heap_sz*)
  - Query Heap Size (*query_heap_sz*)

- Agent Stack Size (*agent_stack_sz*)

Agent/Application Shared Memory

- The total number of agent/application shared memory segments for local clients is limited by the lower of the following database configuration parameters:
  - The total of *maxappls* for all active databases
  - The value of *maxagents*, or *max_coordagents* for parallel systems.
- Agent/Application Shared Memory is also affected by the following database configuration parameters:
  - The Application Support Layer Heap Size (*aslheapsz*)parameter
  - The Client I/O Block Size (*rqrioblk*) parameter

Related concepts:

- "Buffer pool management" on page 112
- "Memory allocation in DB2" on page 247
- "Tuning memory allocation parameters" on page 253

The FCM buffer pool and memory requirements

In a partitioned database system, the Database Manager Shared Memory and FCM Buffer Pool are as shown below.

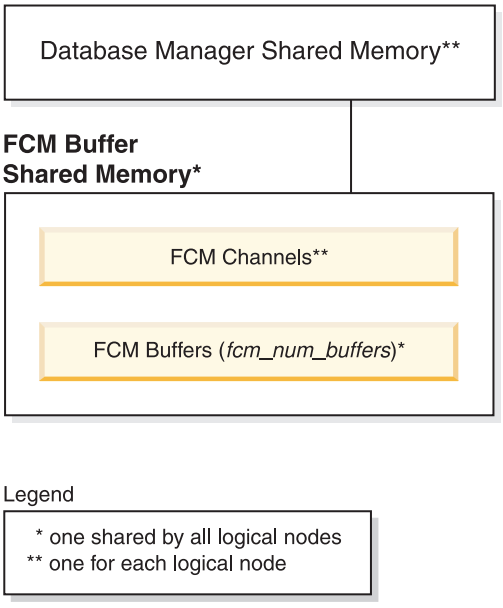


Figure 29. FCM buffer pool when multiple logical nodes are used

The number of FCM buffers for each database partition is controlled by the *fcm_num_buffers* configuration parameter. By default, this parameter is set to AUTOMATIC. To tune this parameter manually, use the data from the buff_free - Buffers currently free and buff_free_bottom - Minimum buffers free system monitor elements.

The number of FCM channels for each database partition is controlled by the *fcm_num_channels* configuration parameter. By default, this parameter is set to

AUTOMATIC. To manually tune this parameter manually, use the data from the ch_free - Channels currently free and ch_free_bottom - Minimum channels free system monitor elements.

Related concepts:

- "Database manager shared memory" on page 249

Related reference:

- "buff_free - FCM Buffers Currently Free monitor element" in *System Monitor Guide and Reference*
- "buff_free_bottom - Minimum FCM Buffers Free monitor element" in *System Monitor Guide and Reference*
- "ch_free - Channels Currently Free monitor element" in *System Monitor Guide and Reference*
- "ch_free_bottom - Minimum Channels Free monitor element" in *System Monitor Guide and Reference*
- "fcm_num_buffers - Number of FCM buffers " on page 487
- "fcm_num_channels - Number of FCM channels " on page 488

Tuning memory allocation parameters

The first rule for setting memory-allocation parameters is never to set them at their highest values unless such a value has been carefully justified. This rule applies even to systems with the maximum amount of memory. Many parameters that affect memory can allow the database manager easily and quickly to take up all of the available memory on a computer. In addition, managing large amounts of memory requires additional work on the part of the database manager and thus incurs more overhead.

Some UNIX operating systems allocate swap space when a process allocates memory and not when a process is paged out to swap space. For these systems, make sure that you provide as much paging space as total shared memory space.

For most configuration parameters, memory is only committed as it is required and the parameter settings determine the maximum size of a particular memory heap. In the following cases, however, the full amount of memory specified by the parameter is allocated:

- Maximum Storage for Lock List (*locklist*)
- Application Support Layer Heap Size (*aslheapsz*)
- Number of FCM Buffers (*fcm_num_buffers*)
- Number of FCM Channels (*fcm_num_channels*)

Parameters that affect application-group memory use

The parameters that affect application-group use of memory apply only to partitioned databases, databases for which intra-parallel processing is enabled, and databases for which the connection concentrator is enabled. The following parameters determine how applications in application groups use their shared memory:

- The *appgroup_mem_sz* parameter specifies the size of the shared memory for the application group.

Setting the *appgroup_mem_sz* configuration parameter too high has an adverse effect. Because all applications in the application group share the caches in the application-group heap, having too many applications will increase cache contention. On the other hand, if each application group contains few applications, the effect of the cache is also limited.

- The *groupheap_ratio* parameter specifies the percent of memory allowed for the shared heap.

  Setting *groupheap_ratio* too low limits the size of caches. Setting the *groupheap_ratio* too high causes the application control heap to be too small and might cause SQL error `SQL0973`, which warns you that you are running out of application control-heap memory at run time.
- The *app_ctl_heap_sz* parameter specifies the size of the control heap for each application in the group.

Accept the default setting for these parameters when you configure your database server. Adjust the settings only if performance suffers. For example, set *appgroup_mem_sz* to control the number of applications in each application group. As a rule of thumb, consider that 10 is too small and 100 is too many. The default is probably appropriate. Then run an average workload and use the Health Center utility from the Control Center or the system monitor to collect information about the hit ratios for the catalog cache, the package cache, and the shared workspace.

- If many `sql0973` errors occur, the *groupheap_ratio* setting is too high.
- If monitoring data shows that the average and maximum usage of the application control heap is far below `app_ctl_heap_sz * (100 - groupheap_ratio)/ 100`, reduce the value of the *app_ctl_heap_sz* configuration parameter.
- If the cache usage indicates that the caches are reaching their limit, increase the value of the *group_heap_ratio* configuration parameter or reduce the number of applications in the application group.

Notes:

- Benchmark tests provide the best information about setting appropriate values for memory parameters. In benchmarking, typical and worst-case SQL statements are run against the server and the values of the parameters are modified until the point of diminishing return for performance is found. If performance versus parameter value is graphed, the point at which the curve begins to plateau or decline indicates the point at which additional allocation provides no additional value to the application and is therefore simply wasting memory.
- The upper limits of memory allocation for several parameters may be beyond the memory capabilities of existing hardware and operating systems. These limits allow for future growth.
- For valid parameter ranges, refer to the detailed information about each parameter.

Related concepts:

- “Database manager shared memory” on page 249
- “Memory allocation in DB2” on page 247

Related reference:

- “app_ctl_heap_sz - Application control heap size ” on page 392
- “appgroup_mem_sz - Maximum size of application group memory set ” on page 394

- “aslheapsz - Application support layer heap size ” on page 403
- “catalogcache_sz - Catalog cache size ” on page 378
- “dbheap - Database heap ” on page 382
- “fcm_num_buffers - Number of FCM buffers ” on page 487
- “groupheap_ratio - Percent of memory for application group heap ” on page 395
- “locklist - Maximum storage for lock list ” on page 383
- “pckcachesz - Package cache size ” on page 387
- “sheapthres - Sort heap threshold ” on page 399
- “util_heap_sz - Utility heap size ” on page 392

Automatic configuration using self tuning memory

Self tuning memory

Starting in IBM DB2 Version 9, a new memory tuning feature simplifies the task of memory configuration by automatically setting values for several memory configuration parameters. When enabled, the memory tuner dynamically distributes available memory resources among several memory consumers including sort, the package cache, the lock list and buffer pools.

The memory tuner is responsive to significant changes in workload characteristics, adjusting the values of memory configuration parameters and the sizes of buffer pools to optimize performance.

The tuner works within the memory limits defined by the *database_memory* configuration parameter. The value of *database_memory* can itself be automatically tuned on Windows and AIX. When self-tuning is enabled for *database_memory* (when it is set to AUTOMATIC), the tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory depending on the current database requirements. For example, if the current database requirements are high, and there is sufficient free memory on the system, more memory will be consumed by database shared memory. Once the database memory requirements drop, or the amount of free memory on the system drops too low, some database shared memory is released.

When the *database_memory* parameter is not enabled for self tuning (when it is not set to AUTOMATIC), the entire database will use the specified amount of memory, distributing it across the database memory consumers as required. When database_memory is not enabled for self tuning, the amount of memory used by the database can be specified in two ways: setting database_memory to a numeric value or setting it to COMPUTED. In the second case, the total amount of memory is computed based on the sum of the initial values of the database memory heaps at database startup time.

The following memory consumers can be enabled for self tuning:

- Buffer pools (controlled by the ALTER BUFFERPOOL and CREATE BUFFERPOOL statements)
- Package cache (controlled by the *pckcachesz* configuration parameter)
- Locking memory (controlled by the *locklist* and *maxlocks* configuration parameters)

- Sort memory (controlled by the *sheapthres_shr* and the *sortheap* configuration parameter)
- Database shared memory (controlled by the *database_memory* configuration parameter)

Related concepts:

- "Using self tuning memory in partitioned database environments" on page 261

Related tasks:

- "Configuring health indicators using a client application" in *System Monitor Guide and Reference*
- "Configuring health indicators using the Health Center" in *System Monitor Guide and Reference*
- "Determining which memory consumers are enabled for self tuning" on page 258
- "Enabling self tuning memory" on page 256

Enabling self tuning memory

Self tuning memory simplifies the task of memory configuration by automatically setting values for memory configuration parameters and sizing buffer pools. When enabled, the memory tuner dynamically distributes available memory resources between several memory consumers including sort, package cache and lock list areas and buffer pools.

Procedure:

1. Enable self tuning for the database by setting *self_tuning_mem* to ON. You can set *self_tuning_mem* to ON using the UPDATE DATABASE CONFIGURATION command, the SQLFUPD API, or through the **Change Database Configuration Parameter** window in the Control Center.
2. To enable self tuning of memory areas controlled by memory configuration parameters, set the relevant configuration parameters to AUTOMATIC using the UPDATE DATABASE CONFIGURATION command, the SQLFUPD API, or through the **Change Database Configuration Parameter** window in the Control Center.
3. To enable self tuning of buffer pools, set the buffer pool size to AUTOMATIC. You can do this using the ALTER BUFFER POOL statement for existing buffer pools or the CREATE BUFFER POOL statement for new buffer pools. If the size of a buffer pool is set to AUTOMATIC in the DPF environment, that buffer pool should not have any entries defined in sysibm.sysbufferpoolnodes.

Notes:

1. Because self tuning redistributes memory between different memory areas, there must be at least two memory areas enabled for self tuning to occur, for example the lock memory area and the database shared memory area. The only exception to this is the memory controlled by the *sortheap* configuration parameter. When *sortheap* alone is set to AUTOMATIC, self tuning of *sortheap* is enabled.
2. In order to enable the *locklist* configuration parameter for self tuning, *maxlocks* must also be enabled for self tuning, therefore *maxlocks* is set to AUTOMATIC when *locklist* is set to AUTOMATIC. In order to enable the *sheapthres_shr*

configuration parameter for self tuning, *sortheap* must also be enabled for self tuning, therefore *sortheap* is set to AUTOMATIC when *sheapthres_shr* is set to AUTOMATIC.

3. Automatic tuning of *sheapthres_shr* or *sortheap* is allowed only when the database manager configuration parameter *sheapthres* is set to 0.
4. Self tuning memory runs only on the HADR primary server. When self tuning memory is activated on an HADR system, it will never run on the secondary server and it will run on the primary server only if the configuration is set properly. If a command is run that switches the HADR database roles, self tuning memory operations will also switch so that they run on the primary server.

Related concepts:

- "Self tuning memory" on page 255

Related tasks:

- "Determining which memory consumers are enabled for self tuning" on page 258
- "Disabling self tuning memory" on page 257

Related reference:

- "self_tuning_mem- Self tuning memory " on page 389
- "sqlfupd data structure" in *Administrative API Reference*

Disabling self tuning memory

Self tuning can be disabled for the entire database by setting *self_tuning_mem* to OFF. When *self_tuning_mem* is set to OFF, the memory configuration parameters and buffer pools that are set to AUTOMATIC remain AUTOMATIC and the memory areas remain at their current size.

You can set *self_tuning_mem* to OFF using the UPDATE DATABASE CONFIGURATION command, the SQLFUPD API, or through the **Change Database Configuration Parameter** window in the Control Center.

Self tuning can also be effectively deactivated for the entire database if only a single memory consumer is enabled for self tuning. This is because memory cannot be redistributed when only one memory area is enabled.

For example, to disable self tuning of the *sortheap* configuration parameter, you could enter the following:

```
UPDATE DATABASE CONFIGURATION USING SORTHEAP MANUAL
```

To disable self tuning of the *sortheap* configuration parameter and change the current value of *sortheap* to 2000 at the same time, enter the following:

```
UPDATE DATABASE CONFIGURATION USING SORTHEAP 2000
```

In some cases, one memory configuration parameter can only be enabled for self tuning if another related memory configuration parameter is also enabled. For example, self tuning of the *maxlocks* configuration parameter is only permitted when the *locklist* configuration parameter is also enabled. Similarly, self tuning of the *sheapthres_shr* configuration parameter can only be enabled if self tuning of the

sortheap configuration parameter is also enabled. This means that disabling self tuning of the *locklist* or *sortheap* parameters disables self tuning of the *maxlocks* or *sheapthres_shr* parameters, respectively.

Self tuning can be disabled for a buffer pool by setting the buffer pool to MANUAL or setting it to a specific size. For example, both of the following statements will disable self tuning for `bufferpool1`:

```
ALTER BUFFERPOOL bufferpool1 SIZE MANUAL
ALTER BUFFERPOOL bufferpool1 SIZE 1000
```

Related concepts:

- "Self tuning memory" on page 255
- "Using self tuning memory in partitioned database environments" on page 261

Related tasks:

- "Enabling self tuning memory" on page 256

Related reference:

- "self_tuning_mem- Self tuning memory " on page 389

Determining which memory consumers are enabled for self tuning

To view the self tuning settings for memory consumers controlled by configuration parameters, use one of the following methods.

- To view the self tuning settings for configuration parameters from the command line, use the GET DATABASE CONFIGURATION command specifying the SHOW DETAIL parameter.

  The memory consumers that can be enabled for self tuning are grouped together in the output as follows:

```
 Description                        Parameter         Current Value       Delayed Value
 ---------------------------------------------------------------------------------------
 Self tuning memory                    (SELF_TUNING_MEM) = ON (Active)        ON
 Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC(37200)   AUTOMATIC(37200)
 Max storage for lock list (4KB)              (LOCKLIST) = AUTOMATIC(7456)    AUTOMATIC(7456)
 Percent. of lock lists per application       (MAXLOCKS) = AUTOMATIC(98)      AUTOMATIC(98)
 Package cache size (4KB)                  (PCKCACHESZ) = AUTOMATIC(5600)    AUTOMATIC(5600)
 Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = AUTOMATIC(5000)    AUTOMATIC(5000)
 Sort list heap (4KB)                         (SORTHEAP) = AUTOMATIC(256)     AUTOMATIC(256)
```

- You can also use the db2CfgGet API to determine whether or not tuning is enabled. The following values are returned:

```
SQLF_OFF                  0
SQLF_ON_ACTIVE            2
SQLF_ON_INACTIVE          3
```

  `SQLF_ON_ACTIVE` describes a situation where self tuning is enabled and active while `SQLF_ON_INACTIVE` indicates that self tuning is enabled but is currently inactive.

- You can also view the configuration settings in the **Database Configuration** window in the Control Center.

To view the self tuning settings for buffer pools, use one of the following methods.

- To retrieve the list of buffer pools that are enabled for self tuning from the command line, enter:

```
db2 "select BPNAME, NPAGES from sysibm.sysbufferpools"
```

When self tuning is enabled for a buffer pool, the NPAGES field in the sysibm.sysbufferpools table for that particular buffer pool will be set to -2. When self tuning is disabled, the NPAGES field will be set to the buffer pool's current size.

- To determine the current size of buffer pools that have been enabled for self tuning, use the snapshot monitor as follows and examine the bp_cur_buffsz (Current size of buffer pool) monitor element data:

  ```
  db2 get snapshot for bufferpools on db_name
  ```

- To view the self tuning settings of your buffer pools using the Control Center, right-click on a buffer pool and view the attributes of the buffer pools in the object details pane.

It is important to note that responsiveness of the memory tuner is limited by the time required to resize a memory consumer. For example, reducing the size of a buffer pool can be a lengthy process and therefore the performance benefits of trading buffer pool memory for sort area memory may not be immediately realized.

Related concepts:

- "Self tuning memory" on page 255

Related tasks:

- "Enabling self tuning memory" on page 256
- "Disabling self tuning memory" on page 257

Related reference:

- "db2CfgGet API - Get the database manager or database configuration parameters" in *Administrative API Reference*
- "db2GetAlertCfg API - Get the alert configuration settings for the health indicators" in *Administrative API Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "bp_cur_buffsz - Current Size of Buffer Pool monitor element" in *System Monitor Guide and Reference*

Self tuning memory in partitioned database environments

When using the self tuning memory feature in partitioned database environments, there are a few factors that determine whether the feature will tune the system appropriately.

When self tuning memory is enabled in partitioned databases, a single database partition is designated as the tuning partition, and all memory tuning decisions are based on the memory and workload characteristics of that database partition. Once tuning decisions are made on the tuning partition, the memory adjustments are distributed to all other database partitions to ensure that all database partitions maintain similar configurations.

The single tuning partition model necessitates that the feature be used only on database partitions that have similar memory requirements. The following are guidelines to use when determining whether to enable self tuning memory on your partitioned database.

Cases where self tuning is recommended in partitioned databases

When all database partitions have similar memory requirements and are running on similar hardware, self tuning memory can be enabled without any modifications. These types of environments share the following characteristics:

- All database partitions on identical hardware, including an even distribution of multiple logical nodes to multiple physical nodes
- Perfect or near-perfect distribution of data
- Workload running on the database partitions is distributed evenly across database partitions. This means that no one database partition has elevated memory requirements for one or more heaps.

In such an environment, it is desirable to have all database partitions configured equally, and self tuning memory will properly configure the system.

Cases where self tuning is recommended in partitioned databases with care

In cases where most of the database partitions in an environment have similar memory requirements and are running on similar hardware, it is possible to use self tuning memory as long as some care is taken with the initial configuration. These system might have a set of database partitions for data, and a much smaller set of coordinator partitions and a catalog partitions. In such environments, it might be beneficial to configure the coordinator partitions and catalog partitions differently than the database partitions that contain your data.

In this environment, it is still possible to benefit from the self tuning memory feature with some minor setup. Since the database partitions containing the data comprise the bulk of the database partitions, self tuning should be enabled on all of these database partitions and one of these database partitions should be specified as the tuning partition. Additionally, since the catalog and coordinator partitions might be configured differently, self tuning memory should be disabled on these partitions. To disable self tuning on the catalog and coordinator partitions, update the *self_tuning_mem* database configuration parameter on these partitions to OFF.

Cases where self tuning is not recommended in partitioned databases

In environments where the memory requirements of each database partition are different or when different database partitions are running on dramatically different hardware, it is advisable to disable the self tuning memory feature. This can be done by setting the *self_tuning_mem* database configuration parameter to OFF on all partitions.

Comparing memory requirements of different database partitions

The best way to determine if the memory requirements of different database partitions are sufficiently similar is to consult the snapshot monitor. If the following snapshot elements are similar on all partitions (differing by no more than 20%), then the partitions can be considered similar.

Collect the following data by issuing the command `get snapshot for database on <dbname>`.

```
Total Shared Sort heap allocated          = 0
Shared Sort heap high water mark          = 0
Post threshold sorts (shared memory)      = 0
Sort overflows                            = 0
```

```
Package cache lookups                       = 13
Package cache inserts                       = 1
Package cache overflows                     = 0
Package cache high water mark (Bytes)       = 655360

Number of hash joins                        = 0
Number of hash loops                        = 0
Number of hash join overflows               = 0
Number of small hash join overflows         = 0
Post threshold hash joins (shared memory)   = 0

Locks held currently                        = 0
Lock waits                                  = 0
Time database waited on locks (ms)          = 0
Lock list memory in use (Bytes)             = 4968
Lock escalations                            = 0
Exclusive lock escalations                  = 0
```

Collect the following data by issuing the command `get snapshot for bufferpools on <dbname>`

```
Buffer pool data logical reads              = 0
Buffer pool data physical reads             = 0
Buffer pool index logical reads             = 0
Buffer pool index physical reads            = 0
Total buffer pool read time (milliseconds) = 0
Total buffer pool write time (milliseconds)= 0
```

Related concepts:

- "Self tuning memory" on page 255
- "Using self tuning memory in partitioned database environments" on page 261

Related reference:

- "self_tuning_mem- Self tuning memory " on page 389
- "GET SNAPSHOT command" in *Command Reference*

Using self tuning memory in partitioned database environments

When self tuning is enabled in partitioned database environments, there is a single database partition, known as the *tuning partition*, that monitors the memory configuration and propagates any configuration changes to all other database partitions to maintain a consistent configuration across all the database partitions.

The tuning partition is selected based on a number of characteristics, such as the number of database partitions in the partition group and the number of buffer pools defined.

- To determine which database partition is currently specified as the tuning partition, use the following ADMIN_CMD:

  ```
  CALL SYSPROC.ADMIN_CMD( 'get stmm tuning dbpartitionnum' )
  ```

- To change the tuning partition, use the following ADMIN_CMD:

  ```
  CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum <db_partition_num>')
  ```

  When you issue this command, the tuning partition will be updated asynchronously or at the next database startup.

- To have the memory tuner automatically re-select the tuning partition, enter -1 for the <db_partition_num> value.

Starting the memory tuner on DPF systems

The memory tuner will only be started in a DPF environment if the database is activated by an explicit ACTIVATE DATABASE command because self tuning requires all partitions to be active before it can properly tune the memory on a multi-partition system.

Disabling self tuning for a given database partition

- To disable self tuning for a subset of database partitions, set the *self_tuning_mem* configuration parameter to OFF for the database partitions you want to leave untuned.
- To disable self tuning for a subset of the memory consumers controlled by configuration parameters on a particular database partition, set the value of the relevant configuration parameter or buffer pool size to MANUAL or a specific value on that database partition.
- To disable tuning for a particular buffer pool on a database partition, issue an ALTER BUFFER POOL command specifying a size value and a value for the PARTITIONNUM parameter for the partition where self tuning is to be disabled.

  This ALTER BUFFERPOOL command will create an exception entry for the given buffer pool in the SYSIBM.SYSBUFFERPOOLNODES table. When an entry exists for a buffer pool in this table, that buffer pool can not be re-enabled for self tuning. To remove an exception entry so that a buffer pool can be re-enabled for self tuning:

  1. Issue an ALTER BUFFERPOOL statement setting the buffer pool size to MANUAL.
  2. Issue another ALTER BUFFERPOOL statement setting the buffer pool size on the partition with the exception entry to the same value as the size specified in the SYSIBM.SYSBUFFERPOOLS table.
  3. Enable self tuning by issuing another ALTER BUFFERPOOL statement setting the size to AUTOMATIC.

Enabling self tuning memory in non-uniform environments

Ideally, your data should be distributed evenly across all of your database partitions and the workload run on each partition should have similar memory requirements. If the data distribution is skewed so that one or more of your database partitions contain significantly more or less data than other database partitions, these anomalous database partitions should not be enabled for self tuning. The same is true if the memory requirements are skewed across the database partitions, which can happen, for example, if resource-intensive sorts are only performed on one partition, or if some database partitions are associated with different hardware and more available memory than others. Self tuning can still be enabled on some database partitions in this type of environment. To take advantage of self tuning memory in environments with skew, identify a set of database partitions that have similar data and memory requirements and enable them for self tuning. Memory configuration in the remaining partitions should be configured manually.

Related concepts:

- "Self tuning memory" on page 255

Related tasks:

- "Determining which memory consumers are enabled for self tuning" on page 258

Related reference:

- "self_tuning_mem- Self tuning memory " on page 389

Self tuning memory operational details and limitations

Determining tuning requirements

In order to ensure a fair and relevant comparison between memory consumers, a new common metric has been developed. Each tuned memory consumer calculates the predicted benefit from additional memory, and reports this to the self tuning memory process. Self tuning memory uses these figures as the basis for memory tuning, taking memory from consumers with the least need and reallocating it to those memory areas that will benefit the most.

Frequency of memory tuning

When enabled, self tuning memory will periodically check the variability of database workload. If the workload is not constant (that is, if the queries being run do not exhibit similar memory characteristics), the memory tuner will reallocate memory less frequently - up to 10 minutes between tuning cycles - to achieve more stable trend prediction. For workloads with more constant memory profiles, the memory tuner will tune memory more frequently - as little as 30 seconds between tuning cycles - in order to converge more quickly.

Tracking the progress of self tuning memory

Your current memory configuration can be obtained using the GET DATABASE CONFIGURATION command, or using a snapshot. Changes made by self tuning are recorded in the memory tuning log files in the stmmlog directory. The memory tuning log files contain summaries of the resource demands for each memory consumer at each tuning interval. These intervals can be determined based on the timestamps in the log entries.

Expected time to converge on best configuration

Leaving this feature enabled should result in quick tuning of parameters to optimize memory usage. A system can be tuned from an initial configuration in as little as one hour. In most cases, tuning will usually be complete in at most 10 hours. This worst case occurs when queries run against the database exhibit markedly different memory characteristics.

Limitations of self tuning memory

In cases where low amounts of memory are available (for example, because the value of *database_memory* is set very low, or because multiple databases, instances or other applications are running on the server) performance benefits of self tuning memory will be limited.

Because self tuning memory bases tuning decisions on database workload, workloads with changing memory characteristics limit the ability of self tuning memory to tune effectively. If your workload's memory characteristics are constantly changing, self tuning memory will tune memory less frequently, and will repeatedly tune towards shifting target conditions. In this case, self tuning memory will not achieve absolute convergence, but will instead try to maintain a memory configuration that is tuned to the current workload.

Related concepts:

- "Self tuning memory" on page 255

Related tasks:

- "Disabling self tuning memory" on page 257
- "Enabling self tuning memory" on page 256

Related reference:
- "database_memory - Database shared memory size " on page 380
- "GET DATABASE CONFIGURATION command" in *Command Reference*

Configuration parameters summary

Database Manager Configuration Parameter Summary

The following table lists the parameters in the database manager configuration file for database servers. When changing the database manager configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

For some database manager configuration parameters, the database manager must be stopped (`db2stop`) and restarted (`db2start`) for the new parameter values to take effect. Other parameters can be changed online; these are called *configurable online configuration parameters*. If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command applies the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

The column "Auto." in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DATABASE MANAGER CONFIGURATION command. If you set a parameter to automatic, DB2 will automatically adjust the parameter to reflect current resource requirements.

The column "Perf. Impact" provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.
- **High** — indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — indicates the the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns "Token", "Token Value", and "Data Type" provide information that you will need when calling the **db2CfgGet** or the **db2CfgSet** API. This information includes configuration parameter identifiers, entries for the *token* element in the *db2CfgParam* data structure, and data types for values that are passed to the structure.

Table 62. Configurable Database Manager Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
agent_stack_sz	No	No	Low	SQLF_KTN_AGENT_STACK_SZ	61	Uint16	"agent_stack_sz - Agent stack size " on page 395
agentpri	No	No	High	SQLF_KTN_AGENTPRI	26	Sint16	"agentpri - Priority of agents " on page 422
aslheapsz	No	No	High	SQLF_KTN_ASLHEAPSZ	15	Uint32	"aslheapsz - Application support layer heap size " on page 403
audit_buf_sz	No	No	High	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32	"audit_buf_sz - Audit buffer size " on page 407
authentication[1]	No	No	Low	SQLF_KTN_AUTHENTICATION	78	Uint16	"authentication - Authentication type " on page 506
catalog_noauth	Yes	No	None	SQLF_KTN_CATALOG_NOAUTH	314	Uint16	"catalog_noauth - Cataloging allowed without authority " on page 508
clnt_krb_plugin	No	No	None	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)	"clnt_krb_plugin - Client Kerberos plug-in " on page 509
clnt_pw_plugin	No	No	None	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)	"clnt_pw_plugin - Client userid-password plug-in " on page 509
comm_bandwidth	Yes	No	Medium	SQLF_KTN_COMM_BANDWIDTH	307	float	"comm_bandwidth - Communications bandwidth " on page 498
conn_elapse	Yes	No	Medium	SQLF_KTN_CONN_ELAPSE	508	Uint16	"conn_elapse - Connection elapse time " on page 487
cpuspeed	Yes	No	Low[2]	SQLF_KTN_CPUSPEED	42	float	"cpuspeed - CPU speed " on page 499
dft_account_str	Yes	No	None	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)	"dft_account_str - Default charge-back account " on page 500
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_timestamp* • *dft_mon_uow*	Yes	No	Medium	SQLF_KTN_DFT_MONSWITCHES[3] • SQLF_KTN_DFT_MON_BUFPOOL • SQLF_KTN_DFT_MON_LOCK • SQLF_KTN_DFT_MON_SORT • SQLF_KTN_DFT_MON_STMT • SQLF_KTN_DFT_MON_TABLE • SQLF_KTN_DFT_MON_ TIMESTAMP • SQLF_KTN_DFT_MON_UOW	29 • 33 • 34 • 35 • 31 • 32 • 36 • 30	Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16	"dft_monswitches - Default database system monitor switches " on page 497
dftdbpath	Yes	No	None	SQLF_KTN_DFTDBPATH	27	char(215)	"dftdbpath - Default database path " on page 510
diaglevel	Yes	No	Low	SQLF_KTN_DIAGLEVEL	64	Uint16	"diaglevel - Diagnostic error capture level " on page 493
diagpath	Yes	No	None	SQLF_KTN_DIAGPATH	65	char(215)	"diagpath - Diagnostic data directory path " on page 494
dir_cache	No	No	Medium	SQLF_KTN_DIR_CACHE	40	Uint16	"dir_cache - Directory cache support " on page 408
discover[4]	No	No	Medium	SQLF_KTN_DISCOVER	304	Uint16	"discover - Discovery mode " on page 485
discover_inst	Yes	No	Low	SQLF_KTN_DISCOVER_INST	308	Uint16	"discover_inst - Discover server instance " on page 486
fcm_num_buffers	Yes	Yes	Medium	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32	"fcm_num_buffers - Number of FCM buffers " on page 487
fcm_num_channels	Yes	Yes	Medium	SQLF_KTN_FCM_NUM_CHANNELS	902	Uint32	"fcm_num_channels - Number of FCM channels " on page 488
fed_noauth	Yes	No	None	SQLF_KTN_FED_NOAUTH	806	Uint16	"fed_noauth - Bypass federated authentication " on page 510
federated	No	No	Medium	SQLF_KTN_FEDERATED	604	Sint16	"federated - Federated database system support " on page 501
fenced_pool	No	No	Medium	SQLF_KTN_FENCED_POOL	80	Sint32	"fenced_pool - Maximum number of fenced processes " on page 432
group_plugin	No	No	None	SQLF_KTN_GROUP_PLUGIN	810	char(33)	"group_plugin - Group plug-in " on page 511
health_mon	Yes	No	Low	SQLF_KTN_HEALTH_MON	804	Uint16	"health_mon - Health monitoring " on page 495
indexrec[5]	Yes	No	Medium	SQLF_KTN_INDEXREC	20	Uint16	"indexrec - Index re-creation time " on page 459
instance_memory	No	Yes	Medium	SQLF_KTN_INSTANCE_MEMORY	803	Uint64	"instance_memory - Instance memory " on page 410
intra_parallel	No	No	High	SQLF_KTN_INTRA_PARALLEL	306	Sint16	"intra_parallel - Enable intra-partition parallelism " on page 491
java_heap_sz	No	No	High	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32	"java_heap_sz - Maximum Java interpreter heap size " on page 410
jdk_path	No	No	None	SQLF_KTN_JDK_PATH	311	char(255)	"jdk_path - Software Developer's Kit for Java installation path " on page 501

Table 62. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
keepfenced	No	No	Medium	SQLF_KTN_KEEPFENCED	81	Uint16	"keepfenced - Keep fenced process " on page 433
local_gssplugin	No	No	None	SQLF_KTN_LOCAL_GSSPLUGIN	816	char(33)	"local_gssplugin - GSS API plug-in used for local instance level authorization " on page 512
max_connections	No	No	Medium	SQLF_DBTN_MAX_CONNECTIONS	802	Sint32	"max_connections - Maximum number of client connections " on page 424
max_connretries	Yes	No	Medium	SQLF_KTN_MAX_CONNRETRIES	509	Uint16	"max_connretries - Node connection retries " on page 489
max_coordagents	No	No	Medium	SQLF_KTN_MAX_COORDAGENTS	501	Sint32	"max_coordagents - Maximum number of coordinating agents " on page 425
max_querydegree	Yes	No	High	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32	"max_querydegree - Maximum query degree of parallelism " on page 492
max_time_diff	No	No	Medium	SQLF_KTN_MAX_TIME_DIFF	510	Uint16	"max_time_diff - Maximum time difference among nodes " on page 490
maxagents	No	No	Medium	SQLF_KTN_MAXAGENTS	12	Uint32	"maxagents - Maximum number of agents " on page 426
maxcagents	No	No	Medium	SQLF_KTN_MAXCAGENTS	13	Sint32	"maxcagents - Maximum number of concurrent agents " on page 428
maxtotfilop	No	No	Medium	SQLF_KTN_MAXTOTFILOP	45	Uint16	"maxtotfilop - Maximum total files open " on page 430
mon_heap_sz	No	No	Low	SQLF_KTN_MON_HEAP_SZ	79	Uint16	"mon_heap_sz - Database system monitor heap size " on page 411
nname	No	No	None	SQLF_KTN_NNAME	7	char(8)	"nname - NetBIOS workstation name " on page 483
notifylevel	Yes	No	Low	SQLF_KTN_NOTIFYLEVEL	605	Sint16	"notifylevel - Notify level " on page 496
num_initagents	No	No	Medium	SQLF_KTN_NUM_INITAGENTS	500	Uint32	"num_initagents - Initial number of agents in pool " on page 431
num_initfenced	No	No	Medium	SQLF_KTN_NUM_INITFENCED	601	Sint32	"num_initfenced - Initial number of fenced processes " on page 434
num_poolagents	No	No	High	SQLF_KTN_NUM_POOLAGENTS	502	Sint32	"num_poolagents - Agent pool size " on page 431
numdb	No	No	Low	SQLF_KTN_NUMDB	6	Uint16	"numdb - Maximum number of concurrently active databases including host and iSeries databases " on page 502
query_heap_sz	No	No	Medium	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32	"query_heap_sz - Query heap size " on page 398
resync_interval	No	No	None	SQLF_KTN_RESYNC_INTERVAL	68	Uint16	"resync_interval - Transaction resync interval " on page 465
rqrioblk	No	No	High	SQLF_KTN_RQRIOBLK	1	Uint16	"rqrioblk - Client I/O block size " on page 406
sheapthres	No	No	High	SQLF_KTN_SHEAPTHRES	21	Uint32	"sheapthres - Sort heap threshold " on page 399
spm_log_file_sz	No	No	Low	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32	"spm_log_file_sz - Sync point manager log file size " on page 466
spm_log_path	No	No	Medium	SQLF_KTN_SPM_LOG_PATH	313	char(226)	"spm_log_path - Sync point manager log file path " on page 467
spm_max_resync	No	No	Low	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32	"spm_max_resync - Sync point manager resync agent limit " on page 467
spm_name	No	No	None	SQLF_KTN_SPM_NAME	92	char(8)	"spm_name - Sync point manager name " on page 467
srvcon_auth	No	No	None	SQLF_KTN_SRVCON_AUTH	815	Uint16	"srvcon_auth - Authentication type for incoming connections at the server " on page 512
srvcon_gssplugin_list	No	No	None	SQLF_KTN_SRVCON_GSSPLUGIN_ LIST	814	char(256)	"srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server " on page 513
srv_plugin_mode	No	No	None	SQLF_KTN_SRV_PLUGIN_MODE	809	Uint16	"srv_plugin_mode - Server plug-in mode " on page 514
srvcon_pw_plugin	No	No	None	SQLF_KTN_SRVCON_PW_PLUGIN	813	char(33)	"srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server " on page 513
start_stop_time	Yes	No	Low	SQLF_KTN_START_STOP_TIME	511	Uint16	"start_stop_time - Start and stop timeout " on page 490
svcename	No	No	None	SQLF_KTN_SVCENAME	24	char(14)	"svcename - TCP/IP service name " on page 484
sysadm_group	No	No	None	SQLF_KTN_SYSADM_GROUP	39	char(30)	"sysadm_group - System administration authority group name " on page 514
sysctrl_group	No	No	None	SQLF_KTN_SYSCTRL_GROUP	63	char(30)	"sysctrl_group - System control authority group name " on page 515

Table 62. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
sysmaint_group	No	No	None	SQLF_KTN_SYSMAINT_GROUP	62	char(30)	"sysmaint_group - System maintenance authority group name " on page 516
sysmon_group	No	No	None	SQLF_KTN_SYSMON	808	char(30)	"sysmon_group - System monitor authority group name " on page 517
tm_database	No	No	None	SQLF_KTN_TM_DATABASE	67	char(8)	"tm_database - Transaction manager database name " on page 468
tp_mon_name	No	No	None	SQLF_KTN_TP_MON_NAME	66	char(19)	"tp_mon_name - Transaction processor monitor name " on page 503
trust_allclnts[6]	No	No	None	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16	"trust_allclnts - Trust all clients " on page 518
trust_clntauth	No	No	None	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16	"trust_clntauth - Trusted clients authentication " on page 519
util_impact_lim	Yes	No	High	SQLF_KTN_UTIL_IMPACT_LIM	807	Uint32	"util_impact_lim - Instance impact policy " on page 505

Notes:

1. Valid values (defined in `sqlenv.h`):

   ```
   SQL_AUTHENTICATION_SERVER (0)
   SQL_AUTHENTICATION_CLIENT (1)
   SQL_AUTHENTICATION_DCS (2)
   SQL_AUTHENTICATION_DCE (3)
   SQL_AUTHENTICATION_SVR_ENCRYPT (4)
   SQL_AUTHENTICATION_DCS_ENCRYPT (5)
   SQL_AUTHENTICATION_DCE_SVR_ENC (6)
   SQL_AUTHENTICATION_KERBEROS (7)
   SQL_AUTHENTICATION_KRB_SVR_ENC (8)
   SQL_AUTHENTICATION_GSSPLUGIN (9)
   SQL_AUTHENTICATION_GSS_SVR_ENC (10)
   SQL_AUTHENTICATION_DATAENC (11)
   SQL_AUTHENTICATION_DATAENC_CMP (12)
   SQL_AUTHENTICATION_NOT_SPEC (255)
   ```

2. The *cpuspeed* parameter can have a significant impact on performance, but you should use the default value, except in very specific circumstances, as documented in the parameter description.
3. ```
   Bit 1 (xxxx xxx1): dft_mon_uow
   Bit 2 (xxxx xx1x): dft_mon_stmt
   Bit 3 (xxxx x1xx): dft_mon_table
   Bit 4 (xxxx 1xxx): dft_mon_buffpool
   Bit 5 (xxx1 xxxx): dft_mon_lock
   Bit 6 (xx1x xxxx): dft_mon_sort
   Bit 7 (x1xx xxxx): dft_mon_timestamp
   ```

4. Valid values (defined in `sqlutil.h`):

   ```
   SQLF_DSCVR_KNOWN (1)
   SQLF_DSCVR_SEARCH (2)
   ```

5. Valid values (defined in `sqlutil.h`):

   ```
   SQLF_INX_REC_SYSTEM (0)
   SQLF_INX_REC_REFERENCE (1)
   ```

6. Valid values (defined in `sqlutil.h`):

   ```
   SQLF_TRUST_ALLCLNTS_NO (0)
   SQLF_TRUST_ALLCLNTS_YES (1)
   SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
   ```

Table 63. Informational Database Manager Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
nodetype[1]	SQLF_KTN_NODETYPE	100	Uint16	"nodetype - Machine node type " on page 502
release	SQLF_KTN_RELEASE	101	Uint16	"release - Configuration file release level " on page 472
   ```

Table 63. Informational Database Manager Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information

Notes:
1. Valid values (defined in `sqlutil.h`):
`SQLF_NT_STANDALONE (0)`
`SQLF_NT_SERVER (1)`
`SQLF_NT_REQUESTOR (2)`
`SQLF_NT_STAND_REQ (3)`
`SQLF_NT_MPP (4)`
`SQLF_NT_SATELLITE (5)`				

Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

For some database configuration parameters, changes only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) The changes take effect at the next connection to the database. Other parameters can be changed online; these are called *configurable online configuration parameters*.

The column "Auto." in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DATABASE MANAGER CONFIGURATION command. If you set a parameter to automatic, DB2 will automatically adjust the parameter to reflect current resource requirements.

The column "Perf. Impact" provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates that the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — indicates that the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns "Token", "Token Value", and "Data Type" provide information that you will need when calling the **db2CfgGet** or the **db2CfgSet** API. This information includes configuration parameter identifiers, entries for the *token* element in the *db2CfgParam* data structure, and data types for values that are passed to the structure.

Table 64. Configurable Database Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
alt_collate	No	No	None	SQLF_DBTN_ALT_COLLATE	809	Uint32	"alt_collate - Alternate collating sequence " on page 470
app_ctl_heap_sz	No	No	Medium	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16	"app_ctl_heap_sz - Application control heap size " on page 392
appgroup_mem_sz	No	No	Medium	SQLF_DBTN_APPGROUP_MEM_SZ	800	Uint32	"appgroup_mem_sz - Maximum size of application group memory set " on page 394
applheapsz	No	No	Medium	SQLF_DBTN_APPLHEAPSZ	51	Uint16	"applheapsz - Application heap size " on page 397
archretrydelay	Yes	No	None	SQLF_DBTN_ARCHRETRYDELAY	828	Uint16	"archretrydelay - Archive retry delay on error " on page 445
• *auto_maint* • *auto_db_backup* • *auto_tbl_maint* • *auto_runstats* • *auto_stats_prof* • *auto_prof_upd* • *auto_reorg*	Yes	No	Medium	• SQLF_ENABLE_AUTO_MAINT[1] • SQLF_ENABLE_AUTO_DB_BACKUP • SQLF_ENABLE_AUTO_TBL_MAINT • SQLF_ENABLE_AUTO_RUNSTATS • SQLF_ENABLE_AUTO_STATS_PROF • SQLF_ENABLE_AUTO_PROF_UPD • SQLF_ENABLE_AUTO_REORG	• 831 • 833 • 835 • 837 • 839 • 844 • 841	Uint32	"auto_maint - Automatic maintenance " on page 481
autorestart	Yes	No	Low	SQLF_DBTN_AUTO_RESTART	25	Uint16	"autorestart - Auto restart enable " on page 454
avg_appls	Yes	Yes	High	SQLF_DBTN_AVG_APPLS	47	Uint16	"avg_appls - Average number of active applications " on page 424
blk_log_dsk_ful	Yes	No	None	SQLF_DBTN_BLK_LOG_DSK_FUL	804	Uint16	"blk_log_dsk_ful - Block on log disk full " on page 445
catalogcache_sz	Yes	No	High	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32	"catalogcache_sz - Catalog cache size " on page 378
chngpgs_thresh	No	No	High	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16	"chngpgs_thresh - Changed pages threshold " on page 416
database_memory	No	Yes	Medium	SQLF_DBTN_DATABASE_MEMORY	803	Uint64	"database_memory - Database shared memory size " on page 380
dbheap	Yes	No	Medium	SQLF_DBTN_DB_HEAP	58	Uint64	"dbheap - Database heap " on page 382
db_mem_thresh	Yes	No	Low	SQLF_DBTN_DB_MEM_THRESH	849	Uint16	"db_mem_thresh - Database memory threshold " on page 381
dft_degree	Yes	No	High	SQLF_DBTN_DFT_DEGREE	301	Sint32	"dft_degree - Default degree " on page 475
dft_extent_sz	Yes	No	Medium	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32	"dft_extent_sz - Default extent size of table spaces " on page 417
dft_loadrec_ses	Yes	No	Medium	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16	"dft_loadrec_ses - Default number of load recovery sessions " on page 455
dft_mttb_types	No	No	None	SQLF_DBTN_DFT_MTTB_TYPES	843	Uint32	"dft_mttb_types - Default maintained table types for optimization " on page 476
dft_prefetch_sz	Yes	Yes	Medium	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16	"dft_prefetch_sz - Default prefetch size " on page 418
dft_queryopt	Yes	No	Medium	SQLF_DBTN_DFT_QUERYOPT	57	Sint32	"dft_queryopt - Default query optimization class " on page 476
dft_refresh_age	No	No	Medium	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)	"dft_refresh_age - Default refresh age " on page 477
dft_sqlmathwarn	No	No	None	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16	"dft_sqlmathwarn - Continue upon arithmetic exceptions " on page 477
discover_db	Yes	No	Medium	SQLF_DBTN_DISCOVER	308	Uint16	"discover_db - Discover database " on page 485
dlchktime	Yes	No	Medium	SQLF_DBTN_DLCHKTIME	9	Uint32	"dlchktime - Time interval for checking deadlock " on page 413
dyn_query_mgmt	No	No	Low	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16	"dyn_query_mgmt - Dynamic SQL and XQuery query management " on page 469
failarchpath	Yes	No	None	SQLF_DBTN_FAILARCHPATH	826	char(243)	"failarchpath - Failover log archive path " on page 446
groupheap_ratio	No	No	Medium	SQLF_DBTN_GROUPHEAP_RATIO	801	Uint16	"groupheap_ratio - Percent of memory for application group heap " on page 395
hadr_local_host	No	No	None	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)	"hadr_local_host - HADR local host name " on page 456

Table 64. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
hadr_local_svc	No	No	None	SQLF_DBTN_HADR_LOCAL_SVC	812	char(41)	"hadr_local_svc - HADR local service name " on page 456
hadr_remote_host	No	No	None	SQLF_DBTN_HADR_REMOTE_HOST	813	char(256)	"hadr_remote_host - HADR remote host name " on page 457
hadr_remote_inst	No	No	None	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)	"hadr_remote_inst - HADR instance name of the remote server " on page 457
hadr_remote_svc	No	No	None	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)	"hadr_remote_svc - HADR remote service name " on page 458
hadr_syncmode	No	No	None	SQLF_DBTN_HADR_SYNCMODE	817	Uint32	"hadr_syncmode - HADR synchronization mode for log write in peer state " on page 458
hadr_timeout	No	No	None	SQLF_DBTN_HADR_TIMEOUT	816	Sint32	"hadr_timeout - HADR timeout value " on page 459
indexrec[2]	Yes	No	Medium	SQLF_DBTN_INDEXREC	30	Uint16	"indexrec - Index re-creation time " on page 459
locklist	Yes	Yes	High when it affects escalation	SQLF_DBTN_LOCK_LIST	704	Uint64	"locklist - Maximum storage for lock list " on page 383
locktimeout	No	No	Medium	SQLF_DBTN_LOCKTIMEOUT	34	Sint16	"locktimeout - Lock timeout " on page 413
logarchmeth1	Yes	No	None	SQLF_DBTN_LOGARCHMETH1	822	Uint16	"logarchmeth1 - Primary log archive method " on page 446
logarchmeth2	Yes	No	None	SQLF_DBTN_LOGARCHMETH2	823	Uint16	"logarchmeth2 - Secondary log archive method " on page 447
logarchopt1	Yes	No	None	SQLF_DBTN_LOGARCHOPT1	824	char(243)	"logarchopt1 - Primary log archive options " on page 447
logarchopt2	Yes	No	None	SQLF_DBTN_LOGARCHOPT2	825	char(243)	"logarchopt2 - Secondary log archive options " on page 448
logbufsz	No	No	High	SQLF_DBTN_LOGBUFSZ	33	Uint16	"logbufsz - Log buffer size " on page 386
logfilsiz	No	No	Medium	SQLF_DBTN_LOGFIL_SIZ	92	Uint32	"logfilsiz - Size of log files " on page 436
logindexbuild	Yes	Yes	None	SQLF_DBTN_LOGINDEXBUILD	818	Uint32	"logindexbuild - Log index pages created " on page 448
logprimary	No	No	Medium	SQLF_DBTN_LOGPRIMARY	16	Uint16	"logprimary - Number of primary log files " on page 437
logretain[3]	No	No	Low	SQLF_DBTN_LOG_RETAIN	23	Uint16	"logretain - Log retain enable " on page 449
logsecond	Yes	No	Medium	SQLF_DBTN_LOGSECOND	17	Uint16	"logsecond - Number of secondary log files " on page 439
max_log	Yes	Yes		SQLF_DBTN_MAX_LOG	807	Uint16	"max_log - Maximum log per transaction " on page 440
maxappls	Yes	Yes	Medium	SQLF_DBTN_MAXAPPLS	6	Uint16	"maxappls - Maximum number of active applications " on page 427
maxfilop	Yes	No	Medium	SQLF_DBTN_MAXFILOP	3	Uint16	"maxfilop - Maximum database files open per application " on page 429
maxlocks	Yes	Yes	High when it affects escalation	SQLF_DBTN_MAXLOCKS	15	Uint16	"maxlocks - Maximum percent of lock list before escalation " on page 414
min_dec_div_3	No	No	High	SQLF_DBTN_MIN_DEC_DIV_3	605	Sint32	"min_dec_div_3 - Decimal division scale to 3 " on page 405
mincommit	Yes	No	High	SQLF_DBTN_MINCOMMIT	32	Uint16	"mincommit - Number of commits to group " on page 449
mirrorlogpath	No	No	Low	SQLF_DBTN_MIRRORLOGPATH	806	char(242)	"mirrorlogpath - Mirror log path " on page 440
newlogpath	No	No	Low	SQLF_DBTN_NEWLOGPATH	20	char(242)	"newlogpath - Change the database log path " on page 442
num_db_backups	Yes	No	None	SQLF_DBTN_NUM_DB_BACKUPS	601	Uint16	"num_db_backups - Number of database backups " on page 461
num_freqvalues	Yes	No	Low	SQLF_DBTN_NUM_FREQVALUES	36	Uint16	"num_freqvalues - Number of frequent values retained " on page 479
num_iocleaners	No	Yes	High	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16	"num_iocleaners - Number of asynchronous page cleaners " on page 419
num_ioservers	No	Yes	High	SQLF_DBTN_NUM_IOSERVERS	39	Uint16	"num_ioservers - Number of I/O servers " on page 420

Table 64. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
num_log_span	Yes	Yes		SQLF_DBTN_NUM_LOG_SPAN	808	Uint16	"num_log_span - Number log span " on page 443
num_quantiles	Yes	No	Low	SQLF_DBTN_NUM_QUANTILES	48	Uint16	"num_quantiles - Number of quantiles for columns " on page 480
numarchretry	Yes	No	None	SQLF_DBTN_NUMARCHRETRY	827	Uint16	"numarchretry - Number of retries on error " on page 451
overflowlogpath	No	No	Medium	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)	"overflowlogpath - Overflow log path " on page 443
pckcachesz	Yes	Yes	High	SQLF_DBTN_PCKCACHE_SZ	505	Uint32	"pckcachesz - Package cache size " on page 387
rec_his_retentn	No	No	None	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16	"rec_his_retentn - Recovery history retention period " on page 462
self_tuning_mem	Yes	No	High	SQLF_DBTN_SELF_TUNING_MEM	848	Uint16	"self_tuning_mem- Self tuning memory " on page 389
seqdetect	Yes	No	High	SQLF_DBTN_SEQDETECT	41	Uint16	"seqdetect - Sequential detection flag " on page 421
sheapthres_shr	No	Yes	High	SQLF_DBTN_SHEAPTHRES_SHR	802	Uint32	"sheapthres_shr - Sort heap threshold for shared sorts " on page 390
softmax	No	No	Medium	SQLF_DBTN_SOFTMAX	5	Uint16	"softmax - Recovery range and soft checkpoint interval " on page 451
sortheap	Yes	Yes	High	SQLF_DBTN_SORT_HEAP	52	Uint32	"sortheap - Sort heap size " on page 400
stat_heap_sz	No	No	Low	SQLF_DBTN_STAT_HEAP_SZ	45	Uint32	"stat_heap_sz - Statistics heap size " on page 402
stmtheap	Yes	No	Medium	SQLF_DBTN_STMT_HEAP	821	Uint32	"stmtheap - Statement heap size " on page 402
trackmod	No	No	Low	SQLF_DBTN_TRACKMOD	703	Uint16	"trackmod - Track modified pages enable " on page 463
tsm_mgmtclass	Yes	No	None	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)	"tsm_mgmtclass - Tivoli Storage Manager management class " on page 463
tsm_nodename	Yes	No	None	SQLF_DBTN_TSM_NODENAME	306	char(64)	"tsm_nodename - Tivoli Storage Manager node name " on page 464
tsm_owner	Yes	No	None	SQLF_DBTN_TSM_OWNER	305	char(64)	"tsm_owner - Tivoli Storage Manager owner name " on page 464
tsm_password	Yes	No	None	SQLF_DBTN_TSM_PASSWORD	501	char(64)	"tsm_password - Tivoli Storage Manager password " on page 464
userexit	No	No	Low	SQLF_DBTN_USER_EXIT	24	Uint16	"userexit - User exit enable " on page 452
util_heap_sz	Yes	No	Low	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32	"util_heap_sz - Utility heap size " on page 392
vendoropt	Yes	No	None	SQLF_DBTN_VENDOROPT	829	char(242)	"vendoropt - Vendor options " on page 453

Table 64. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information

Notes:

1. Default =>
```
Bit 1 on  (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x0xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
            0    0     1   9
```

   Maximum =>
```
Bit 1 on  (xxxx xxxx xxxx xxx1): auto_maint
Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd
Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
            0    0    7    F
```

2. Valid values (defined in `sqlutil.h`):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
```

3. Valid values (defined in `sqlutil.h`):

```
SQLF_LOGRETAIN_NO (0)
SQLF_LOGRETAIN_RECOVERY (1)
SQLF_LOGRETAIN_CAPTURE (2)
```

Table 65. Informational Database Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16	"backup_pending - Backup pending indicator " on page 473
codepage	SQLF_DBTN_CODEPAGE	101	Uint16	"codepage - Code page for the database " on page 470
codeset	SQLF_DBTN_CODESET	120	char(9)[1]	"codeset - Codeset for the database " on page 470
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)	"collate_info - Collating information " on page 471
country/region	SQLF_DBTN_COUNTRY	100	Uint16	"country/region - Database territory code " on page 471
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16	"database_consistent - Database is consistent " on page 473
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16	"database_level - Database release level " on page 472
hadr_db_role	SQLF_DBTN_HADR_DB_ROLE	810	Uint32	"hadr_db_role - HADR database role " on page 455
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16	"log_retain_status - Log retain status indicator " on page 473
loghead	SQLF_DBTN_LOGHEAD	105	char(12)	"loghead - First active log file " on page 437

Table 65. Informational Database Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information
logpath	SQLF_DBTN_LOGPATH	103	char(242)	"logpath - Location of log files " on page 437
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16	"multipage_alloc - Multipage file allocation enabled " on page 474
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16	"numsegs - Default number of SMS containers " on page 421
pagesize	SQLF_DBTN_PAGESIZE	846	Uint32	"pagesize - Database default page size" on page 472
release	SQLF_DBTN_RELEASE	102	Uint16	"release - Configuration file release level " on page 472
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16	"restore_pending - Restore pending " on page 474
restrict_access	SQLF_DBTN_RESTRICT_ACCESS	852	Sint32	"restrict_access - Database has restricted access configuration parameter" on page 519
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16	"rollfwd_pending - Roll forward pending indicator " on page 474
territory	SQLF_DBTN_TERRITORY	121	char(5)[2]	"territory - Database territory " on page 472
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16	"user_exit_status - User exit status indicator " on page 475

Notes:

1. char(17) on HP-UX and /.
2. char(33) on HP-UX and /.

DB2 Administration Server (DAS) Configuration Parameter Summary

Table 66. DAS Configuration Parameters

Parameter	Parameter Type	Additional Information
authentication	Configurable	"authentication - Authentication type DAS " on page 520
contact_host	Configurable Online	"contact_host - Location of contact list " on page 521
das_codepage	Configurable Online	"das_codepage - DAS code page " on page 521
das_territory	Configurable Online	"das_territory - DAS territory " on page 522
dasadm_group	Configurable	"dasadm_group - DAS administration authority group name " on page 522
db2system	Configurable Online	"db2system - Name of the DB2 server system " on page 523
discover	Configurable Online	"discover - DAS discovery mode " on page 523

Table 66. DAS Configuration Parameters (continued)

Parameter	Parameter Type	Additional Information
exec_exp_task	Configurable	"exec_exp_task - Execute expired tasks " on page 524
jdk_64_path	Configurable Online	"jdk_64_path - 64-Bit Software Developer's Kit for Java installation path DAS " on page 524
jdk_path	Configurable Online	"jdk_path - Software Developer's Kit for Java installation path DAS " on page 525
sched_enable	Configurable	"sched_enable - Scheduler mode " on page 526
sched_userid	Informational	"sched_userid - Scheduler user ID " on page 526
smtp_server	Configurable Online	"smtp_server - SMTP server " on page 527
toolscat_db	Configurable	"toolscat_db - Tools catalog database " on page 527
toolscat_inst	Configurable	"toolscat_inst - Tools catalog database instance " on page 528
toolscat_schema	Configurable	"toolscat_schema - Tools catalog database schema " on page 528

Part 3. Maintaining Performance

Chapter 12. Collecting statistics

Catalog statistics

When the query compiler optimizes the query plans, its decisions are heavily influenced by statistical information about the size of the database tables, indexes and statistical views. The optimizer also uses information about the distribution of data in specific columns of tables, indexes and statistical views if these columns are used to select rows or join tables. The optimizer uses this information to estimate the costs of alternative access plans for each query.

In addition to table size and data distribution information, you can also collect statistical information about the cluster ratio of indexes, the number of leaf pages in indexes, the number of table rows that overflow their original pages, and the number of filled and empty pages in a table. You use this information to decide when to reorganize tables and indexes.

When you collect statistics for a table in a partitioned database environment, statistics are only collected for that portion of the table that resides on the database partition where the utility is executed, or for the first database partition in the database partition group that contains the table. When you collect statistics for a statistical view, statistics are collected for all database partitions.

When you execute the RUNSTATS utility for a table, statistical view, or for a table and its associated indexes, the following kinds of statistical information are stored in the system catalog tables:

For a table and index:
- The number of pages in use
- The number of pages that contain rows
- The number of rows that overflow
- The number of rows in the table (cardinality)
- For MDC tables, the number of blocks that contain data
- For partitioned tables, the degree of data clustering within a single data partition

For each column in the table or statistical view and the first column in the index key:
- The cardinality of the column
- The average length of the column
- The second highest value in the columns
- The second lowest value in the column
- The number of NULLs in the column

For each XML column, the following statistical information are collected. Each row in an XML column stores an XML document. The node count of a given path or path-value pair refers to the number of nodes reachable by the path or path-value pair. The document count of a given path or path-value pair refers to the number of documents that contains the given path or path-value pair.
- The number of NULL XML documents
- The number of non-NULL XML documents
- The number of distinct paths

- The sum of the node count of each distinct path
- The sum of the document count of each distinct path
- The *k* pairs of (path, node count) with the largest node count
- The *k* pairs of (path, document count) with the largest document count
- The *k* triples of (path, value, node count) with the largest node count
- The *k* triples of (path, value, document count) with the largest document count
- For each distinct path that leads to a text or attribute value:
  - The number of distinct values this path can take
  - The highest value
  - The lowest value
  - The number of text or attribute nodes
  - The number of documents that contain the text or attribute nodes

For groups of columns that you specify:
- A timestamp based name for the column group
- The cardinality of the column group

For indexes only:
- The number of index entries (index cardinality)
- The number of leaf pages
- The number of index levels
- The degree of clustering of the table data to this index
- The degree of clustering of the index keys with regard to data partitions
- The ratio of leaf pages on disk in index key order to the number of pages in the range of pages occupied by the index
- The number of distinct values in the first column of the index
- The number of distinct values in the first two, three, and four columns of the index
- The number of distinct values in all columns of the index
- The number of leaf pages located on disk in index key order, with few or no large gaps between them
- The number of pages on which all RIDs are marked deleted
- The number of RIDs marked deleted on pages on which not all RIDs are marked deleted

If you request detailed statistics for an index, you also store finer information about the degree of clustering of the table to the index and the page fetch estimates for different buffer sizes.

You can also collect the following kinds statistics about tables and indexes:
- Data distribution statistics

  The optimizer uses data distribution statistics to estimate efficient access plans for tables and statistical views in which data is not evenly distributed and columns have a significant number of duplicate values.
- Detailed index statistics

  The optimizer uses detailed index statistics to determine how efficient it is to access a table through an index.
- Sub-element statistics

  The optimizer uses sub-element statistics for LIKE predicates, especially those that search for a pattern embedded within a string, such as `LIKE %disk%`.

Distribution statistics are not collected:

- When the *num_freqvalues* and *num_quantiles* configuration parameters are set to zero (0)
- When the distribution of data is known, such as when each data value is unique.
- When the column is a data type for which statistics are never collected. These data type are LONG, large object (LOB), or structured columns.
- For row types in sub-tables, the table level statistics NPAGES, FPAGES, and OVERFLOW are not collected.
- If quantile distributions are requested, but there is only one non-NULL value in the column
- For extended indexes or declared temporary tables

Note: You can perform a RUNSTATS on a declared temporary table, but the resulting statistics are not stored in the system catalogs because declared temporary tables do not have catalog entries. However, the statistics are stored in memory structures that represent the catalog information for declared temporary tables. In some cases, therefore, it might be useful to perform a RUNSTATS on these tables.

Related concepts:

- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "Distribution statistics" on page 291
- "General rules for updating catalog statistics manually" on page 306
- "Statistical views" on page 179
- "Statistics for modeling production databases" on page 304

Related tasks:

- "Collecting catalog statistics" on page 279

Related reference:

- "RUNSTATS command" in *Command Reference*
- "num_freqvalues - Number of frequent values retained " on page 479
- "num_quantiles - Number of quantiles for columns " on page 480

Collecting catalog statistics

You collect catalog statistics on tables, indexes and statistical views to provide information that the optimizer uses to choose the best access plans for queries.

Prerequisites:

For tables and indexes, you must have one of the following:

- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the table
- LOAD authority

For statistical views, you must have one of the following:

- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the table

In addition, you must have privileges to access rows from the statistical view. Specifically, for each table, view or nickname referenced in the statistical view definition, you must have one of the following privileges:

- SYSADM or DBADM
- CONTROL
- SELECT

Procedure:

To collect catalog statistics:

1. Connect to the database that contains the tables, indexes or statistical views for which you want to collect statistical information.
2. From the DB2 command line, execute the RUNSTATS command with appropriate options. These options allow you to tailor the statistics that are collected for the queries that run against the tables, indexes or statistical views.
3. When RUNSTATS is complete, issue a COMMIT statement to release locks.
4. Rebind packages that access tables, indexes or statistical views for which you have regenerated statistical information.

To use a graphical user interface to specify options and collect statistics, use the Control Center.

Notes:

1.

   Note: Because the RUNSTATS utility does not support use of nicknames, you update statistics differently for federated database queries. If queries access a federated database, execute RUNSTATS for the tables in all databases, then drop and recreate the nicknames that access remote tables to make the new statistics available to the optimizer.

2.

   Note:

   When you collect statistics for a table in a partitioned database environment, RUNSTATS only collects statistics for tables on the database partition from which you execute it. The RUNSTATS results from this database partition are extrapolated to the other database partitions. If the database partition from which you execute RUNSTATS does not contain a portion of a particular table, the request is sent to the first database partition in the database partition group that contains a portion of the table.

   When you collect statistics for a statistical view, statistics are collected for all database partitions containing base tables referenced by the view.

Related concepts:

- "Catalog statistics" on page 277
- "Guidelines for collecting and updating statistics" on page 283
- "Statistical views" on page 179

Related tasks:
- "Collecting distribution statistics for specific columns" on page 281
- "Collecting index statistics" on page 282
- "Determining when to reorganize tables" on page 320

Related reference:
- "RUNSTATS command" in *Command Reference*
- "RUNSTATS command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

Collecting distribution statistics for specific columns

For efficiency both of RUNSTATS and subsequent query-plan analysis, you might collect distribution statistics on only the columns that queries use in WHERE, GROUP BY, and similar clauses. You might also collect cardinality statistics on combined groups of columns. The optimizer uses such information to detect column correlation when it estimates selectivity for queries that reference the columns in the group.

In the following steps, the database is assumed to be **sales** and to contain the table **customers**, with indexes **custidx1** and **custidx2**.

Prerequisites:

You must connect to the database that contains the tables and indexes and have one of the following authorization levels:
- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the table

Note: RUNSTATS only collects statistics for tables on the database partition from which you execute it. The RUNSTATS results from this database partition are extrapolated to the other database partitions. If the database partition from which you execute RUNSTATS does not contain a portion of a table, the request is sent to the first database partition in the database partition group that holds that portion of the table.

Procedure:

To collect statistics on specific columns:
1. Connect to the **sales** database.
2. Execute one of the following commands at the DB2 command line, depending on your requirements:
   - To collect distribution statistics on columns **zip** and **ytdtotal**:

```
RUNSTATS ON TABLE sales.customers
     WITH DISTRIBUTION ON COLUMNS (zip, ytdtotal)
```

- To collect distribution statistics on the same columns, but adjust the distribution defaults:

```
RUNSTATS ON TABLE sales.customers
     WITH DISTRIBUTION ON
     COLUMNS (zip, ytdtotal NUM_FREQVALUES 50 NUM_QUANTILES 75)
```

- To collect distribution statistics on the columns indexed in **custidx1** and **custidx2**:

```
RUNSTATS ON TABLE sales.customer
         ON KEY COLUMNS
```

- To collect column statistics on the table only for specific columns **zip** and **ytdtotal** and a column group that includes **region** and **territory**:

```
RUNSTATS ON TABLE sales.customers
          ON COLUMNS (zip, (region, territory), ytdtotal)
```

- Suppose statistics for non-XML columns have been collected using the LOAD command with the STATISTICS option. To complement the non-XML statistics with the statistics for the XML column **miscinfo**:

```
RUNSTATS ON TABLE sales.customers
   ON COLUMNS (miscinfo)
```

- To collect column statistics on the table for non-XML columns only (the EXCLUDING XML COLUMNS option takes precedence over all other clauses that may specify XML columns):

```
RUNSTATS ON TABLE sales.customers
   EXCLUDING XML COLUMNS
```

You can also use the Control Center to collect distribution statistics.

Related concepts:

- "Catalog statistics tables" on page 286
- "Column correlation for multiple predicates" on page 193
- "Using index and column group statistics to compute grouping keycard" on page 174

Related tasks:

- "Collecting catalog statistics" on page 279
- "Collecting index statistics" on page 282

Collecting index statistics

Collect index statistics to allow the optimizer to evaluate whether an index should be used to resolve a query.

In the following steps, the database is assumed be **sales** and to contain the table **customers**, with indexes **custidx1** and **custidx2**.

Prerequisites:

You must connect to the database that contains the tables and indexes and have one of the following authorization levels:

- sysadm
- sysctrl
- sysmaint
- dbadm

- CONTROL privilege on the table

Executing RUNSTATS with the SAMPLED DETAILED option requires 2MB of the statistics heap. Allocate an additional 488 4K pages to the *stat_heap_sz* database configuration parameter setting for this additional memory requirement. If the heap appears to be too small, RUNSTATS returns an error before it attempts to collect statistics.

Procedure:

To collect detailed statistics for an index:

1. Connect to the **sales** database.
2. Execute one of the following commands at the DB2 command line, depending on your requirements:
   - To create detailed statistics on both **custidx1** and **custidx2**:
     ```
     RUNSTATS ON TABLE sales.customers AND DETAILED INDEXES ALL
     ```
   - To create detailed statistics on both indexes, but use sampling instead of performing detailed calculations for each index entry:
     ```
     RUNSTATS ON TABLE sales.customers AND SAMPLED DETAILED INDEXES ALL
     ```
   - To create detailed sampled statistics on indexes as well as distribution statistics for the table so that index and table statistics are consistent:
     ```
     RUNSTATS ON TABLE sales.customers
              WITH DISTRIBUTION ON KEY COLUMNS
              AND SAMPLED DETAILED INDEXES ALL
     ```

You can also use the Control Center to collect index and table statistics.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics tables" on page 286
- "Detailed index statistics" on page 294

Related tasks:

- "Collecting catalog statistics" on page 279

Guidelines for collecting and updating statistics

The RUNSTATS command collects statistics on tables, indexes and statistical views to provide the optimizer with accurate information for access plan selection.

Use the RUNSTATS utility to collect statistics in the following situations:

- When data has been loaded into a table and the appropriate indexes have been created.
- When you create a new index on a table. You need execute RUNSTATS for only the new index if the table has not been modified since you last ran RUNSTATS on it.
- When a table has been reorganized with the REORG utility.
- When the table and its indexes have been extensively updated, by data modifications, deletions, and insertions. ("Extensive" in this case might mean that 10 to 20 percent of the table and index data has been affected.)
- Before binding application programs whose performance is critical
- When you want to compare current and previous statistics. If you update statistics at regular intervals you can discover performance problems early.

- When the prefetch quantity is changed.
- When you have used the REDISTRIBUTE DATABASE PARTITION GROUP utility.

  Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.
- Use the RUNSTATS utility to collect statistics on XML columns. When RUNSTATS is used to collect statistics for XML columns only, existing statistics for non-XML columns that have been collected by LOAD or a previous execution of the RUNSTATS utility are retained. In the case where statistics on some XML columns have been collected previously, the previously collected statistics for an XML column will either be dropped if no statistics on that XML column are collected by the current command, or be replaced if statistics on that XML column are collected by the current command.

To improve RUNSTATS performance and save disk space used to store statistics, consider specifying only the columns for which data distribution statistics should be collected.

Ideally, you should rebind application programs after running statistics. The query optimizer might choose a different access plan if it has new statistics.

If you do not have enough time to collect all of the statistics at one time, you might run RUNSTATS to update statistics on only a few tables, indexes, or statistical views at a time, rotating through the set of objects. If inconsistencies occur as a result of activity on the table between the periods where you run RUNSTATS with a selective partial update, then a warning message (SQL0437W, reason code 6) is issued during query optimization. For example, this might occur if you execute RUNSTATS to gather table distribution statistics and, after some table activity, execute RUNSTATS again to gather index statistics on that table. If inconsistencies occur as a result of the activity on the table and these inconsistencies are detected during query optimization, the warning message is issued. When this happens, you should run RUNSTATS again to update the distribution statistics.

To ensure that the index statistics are synchronized with the table, execute RUNSTATS to collect both table and index statistics at the same time. Index statistics retain most of the table and column statistics collected from the last run of RUNSTATS. If the table has been modified extensively since the last time its table statistics were gathered, gathering only the index statistics for that table will leave the two sets of statistics out of synchronization on all nodes.

Invoking RUNSTATS on a production system might negatively impact the performance of the production workload. The RUNSTATS utility now supports a throttling option which can be used to limit the performance impact of RUNSTATS execution during high levels of database activity.

When you collect statistics for a table in a partitioned database environment, RUNSTATS only collects statistics for tables on the database partition from which you execute it. The RUNSTATS results from this database partition are extrapolated to the other database partitions. If the database partition from which you execute RUNSTATS does not contain a portion of a particular table, the request is sent to the first database partition in the database partition group that contains a portion of the table.

When you collect statistics for a statistical view, statistics are collected for all database partitions containing base tables referenced by the view.

Consider these tips to improve the efficiency of RUNSTATS and the usefulness of the collected statistics:

- Collect statistics only for the columns used to join tables or in the WHERE, GROUP BY, and similar clauses of queries. If these columns are indexed, you can specify the columns with the ONLY ON KEY COLUMNS clause for the RUNSTATS command.
- Customize the values for *num_freqvalues* and *num_quantiles* for specific tables and specific columns in tables.
- Collect DETAILED index statistics with the SAMPLE DETAILED clause to reduce the amount of background calculation performed for detailed index statistics. The SAMPLE DETAILED clause reduces the time required to collect statistics, and produces adequate precision in most cases.
- When you create an index for a populated table, add the COLLECT STATISTICS clause to create statistics as the index is created.
- When significant numbers of table rows are added or removed, or if data in columns for which you collect statistics is updated, execute RUNSTATS again to update the statistics.
- Since RUNSTATS only collects statistics on a single database partition, the statistics will be less accurate if the data is not distributed consistently across all database partitions. If you suspect that there is skewed data distribution, you might want to redistribute the data across database partitions using the REDISTRIBUTE DATABASE PARTITION GROUP command before executing RUNSTATS.

Related concepts:

- "Automatic statistics collection" on page 311
- "Catalog statistics" on page 277
- "Optimizer use of distribution statistics" on page 298

Related tasks:

- "Collecting catalog statistics" on page 279
- "Collecting distribution statistics for specific columns" on page 281
- "Collecting index statistics" on page 282

Collecting statistics on a sample of the table data

Table statistics are used by the query optimizer in selecting the best access plan for any given query, so it is important that statistics remain current to accurately reflect the state of a table at any given time. As the activity against a table increases, so should the frequency of statistics collection. With the increasing size of databases, it is becoming more important to find efficient ways to collect statistics. Random sampling of table data on which to collect statistics can improve RUNSTATS performance. For I/O bound or CPU bound systems, the performance benefits can be enormous. The smaller the sample, the faster RUNSTATS completes.

Starting in Version 8.2, the RUNSTATS command provides the option to collect statistics on a sample of the data in the table by using the TABLESAMPLE option.

This feature can increase the efficiency of statistics collection since sampling uses only a subset of the data. At the same time, the sampling methods ensure a high level of accuracy.

There are two ways to specify how the sample is to be collected. The BERNOULLI method samples the data at the level of the row. During a full table scan of the data pages each row is considered in turn and is selected based on probability P as specified by the numeric parameter. It is only on these selected rows that statistics will be collected. In a similar manner, the SYSTEM method samples the data at the page-level. Thus, each page is selected on probability P and rejected with probability 1-P/100.

Performance of page-level sampling is excellent because only one I/O is required for each selected page. With row-level sampling, I/O costs are not reduced since every table page is retrieved in a full table scan. However, row-level sampling provides significant improvements, even if the amount of I/O is not reduced, because gathering statistics is CPU intensive.

Row-level sampling will provide a better sample than page-level sampling in situations where the data values are highly clustered. Compared to page sampling, the row-level sample set will likely be a better reflection of the data since it will include P percent rows from each data page. In page-level sampling all the rows of P percent pages will be in the sample set. If the rows are distributed randomly over the table, then the accuracy of row sampled statistics will be similar to the accuracy of page sampled statistics.

Each sample is randomly generated across RUNSTATS commands unless the REPEATABLE option is used. With the REPEATABLE clause, the same sample will be generated as in the last execution of the RUNSTATS command with the TABLESAMPLE option. Users may find this beneficial in cases where the generation of consistent statistics is desired for tables of constant data.

Related concepts:

- "Data sampling in SQL and XQuery queries" on page 109

Related reference:

- "RUNSTATS command" in *Command Reference*

Catalog statistics tables

The following tables provide information about the system catalog tables that contain catalog statistics and the RUNSTATS options that collect specific statistics.

Table 67. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
FPAGES	number of pages being used by a table.	Yes	Yes
NPAGES	number of pages containing rows	Yes	Yes
OVERFLOW	number of rows that overflow	Yes	No

Table 67. Table Statistics (SYSCAT.TABLES and SYSSTAT.TABLES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
CARD	number of rows in table (cardinality)	Yes	Yes (Note 1)
ACTIVE_BLOCKS	for MDC tables, the total number of occupied blocks	Yes	No

Note:
1. If the table has no indexes defined and you request statistics for indexes, no new CARD statistics are updated. The previous CARD statistics are retained.

Table 68. Column Statistics (SYSCAT.COLUMNS and SYSSTAT.COLUMNS)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
COLCARD	column cardinality	Yes	Yes (Note 1)
AVGCOLLEN	average length of column	Yes	Yes (Note 1)
HIGH2KEY	second highest value in column	Yes	Yes (Note 1)
LOW2KEY	second lowest value in column	Yes	Yes (Note 1)
NUMNULLS	the number of NULLs in a column	Yes	Yes (Note 1)
SUB_COUNT	the average number of subelements	Yes	No (Note 2)
SUB_DELIM_LENGTH	average length of each delimiter separating each subelement	Yes	No (Note 2)

Note:
1. Column statistics are gathered for the first column in the index key.
2. These statistics provide information about data in columns that contain a series of subfields or subelements that are delimited by blanks. The SUB_COUNT and SUB_DELIM_LENGTH statistics are collected only for single-byte character set string columns of type CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC.

Table 69. Multicolumn Statistics (SYSCAT.COLGROUPS and SYSSTAT.COLGROUPS)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
COLGROUPCARD	cardinality of the column group	Yes	No

Note: The multicolumn distribution statistics listed in the following two tables are not collected by RUNSTATS. You cannot update them manually.

Table 70. Multicolumn Distribution Statistics (SYSCAT.COLGROUPDIST and SYSSTAT.COLGROUPDIST)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
TYPE	F = frequency value Q = quantile value	Yes	No

Table 70. Multicolumn Distribution Statistics (SYSCAT.COLGROUPDIST and SYSSTAT.COLGROUPDIST) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
ORDINAL	Ordinal number of the column in the group	Yes	No
SEQNO	Sequence number *n* that represents the *n*th TYPE value	Yes	No
COLVALUE	the data value as a character literal or a null value	Yes	No

Table 71. Multicolumn Distribution Statistics 2 (SYSCAT.COLGROUPDISTCOUNTS and SYSSTAT.COLGROUPDISTCOUNTS)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
TYPE	F = frequency value Q = quantile value	Yes	No
SEQNO	Sequence number *n* that represents the *n*th TYPE value	Yes	No
VALCOUNT	If TYPE = F, VALCOUNT is the number of occurrences of COLVALUEs for the column group identified by this SEQNO. If TYPE = Q, VALCOUNT is the number of rows whose value is less than or equal to COLVALUEs for the column group with this SEQNO.	Yes	No
DISTCOUNT	If TYPE = Q, this column contains the number of distinct values that are less than or equal to COLVALUEs for the column group with this SEQNO. Null if unavailable.	Yes	No

Table 72. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
NLEAF	number of index leaf pages	No	Yes
NLEVELS	number of index levels	No	Yes
CLUSTERRATIO	degree of clustering of table data	No	Yes (Note 2)
CLUSTERFACTOR	finer degree of clustering	No	Detailed (Notes 1,2)

Table 72. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	Indexes
DENSITY	Ratio (percentage) of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index (Note 3)	No	Yes
FIRSTKEYCARD	number of distinct values in first column of the index	No	Yes
FIRST2KEYCARD	number of distinct values in first two columns of the index	No	Yes
FIRST3KEYCARD	number of distinct values in first three columns of the index	No	Yes
FIRST4KEYCARD	number of distinct values in first four columns of the index	No	Yes
FULLKEYCARD	number of distinct values in all columns of the index, excluding any key value in a type-2 index for which all RIDs are marked deleted	No	Yes
PAGE_FETCH_PAIRS	page fetch estimates for different buffer sizes	No	Detailed (Notes 1,2)
AVGPARTITION_CLUSTERRATIO	Degree of data clustering within a single data partition	No	Yes (Note 2)
AVGPARTITION_CLUSTERFACTOR	Finer measurement of degree of clustering, within a single data partition.	No	Detailed (Notes 1,2)
AVGPARTITION_PAGE_FETCH_PAIRS	page fetch estimates for different buffer sizes generated based on a single data partition	No	Detailed (Notes 1,2)
DATAPARTITION_CLUSTERFACTOR	Number of data partition references during an index scan	No (Note 6)	Yes (Note 6)
SEQUENTIAL_PAGES	number of leaf pages located on disk in index key order, with few or no large gaps between them	No	Yes
AVERAGE_SEQUENCE_PAGES	average number of index pages accessible in sequence. This is the number of index pages that the prefetchers can detect as being in sequence.	No	Yes
AVERAGE_RANDOM_PAGES	average number of random index pages between sequential page accesses	No	Yes
AVERAGE_SEQUENCE_GAP	gap between sequences	No	Yes
AVERAGE_SEQUENCE_FETCH_PAGES	average number of table pages accessible in sequence. This is the number of table pages that the prefetchers can detect as being in sequence when they fetch table rows using the index.	No	Yes (Note 4)

Table 72. Index Statistics (SYSCAT.INDEXES and SYSSTAT.INDEXES) (continued)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
AVERAGE_RANDOM_FETCH_PAGES	average number of random table pages between sequential page accesses when fetching table rows using the index.	No	Yes (Note 4)
AVERAGE_SEQUENCE_FETCH_GAP	gap between sequences when fetching table rows using the index.	No	Yes (Note 4)
NUMRIDS	the number of record identifiers (RIDs) in the index, including deleted RIDs in type-2 indexes.	No	Yes
NUMRIDS_DELETED	the total number of RIDs marked deleted in the index, except RIDs on leaf pages on which all record identifiers are marked deleted	No	Yes
NUM_EMPTY_LEAFS	the total number of leaf pages on which all record identifiers are marked deleted	No	Yes
INDCARD	Number of index entries (index cardinality)	No	Yes

Note:
1. Detailed index statistics are gathered by specifying the DETAILED clause on the RUNSTATS command.
2. CLUSTERFACTOR and PAGE_FETCH_PAIRS are not collected with the DETAILED clause unless the table is of a respectable size. If the table is greater than about 25 pages, then CLUSTERFACTOR and PAGE_FETCH_PAIRS statistics are collected. In this case, CLUSTERRATIO is -1 (not collected). If the table is a relatively small table, only CLUSTERRATIO is filled in by RUNSTATS while CLUSTERFACTOR and PAGE_FETCH_PAIRS are not. If the DETAILED clause is not specified, only the CLUSTERRATIO statistic is collected.
3. This statistic measures the percentage of pages in the file containing the index that belongs to that table. For a table having only one index defined on it, DENSITY should normally be 100. DENSITY is used by the optimizer to estimate how many irrelevant pages from other indexes might be read, on average, if the index pages were prefetched.
4. These statistics cannot be computed when this table is in a DMS table space.
5. Prefetch statistics will not be collected during a LOAD or CREATE INDEX even if statistics collection is specified when the command is invoked. Prefetch statistics are also not collected if the Sequential Detection Flag configuration parameter (seqdetect) is turned off.
6. When RUNSTATS options for Table is ″No″ it means statistics are not collected when table statistics are collected, and ″Yes″ for Indexes means statistics are collected when the RUNSTATS command is used with the INDEXES options.

Table 73. Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
DISTCOUNT	If TYPE is Q, the number of distinct values that are less than or equal to COLVALUE statistics	Distribution (Note 2)	No
TYPE	Indicator of whether row provides frequent-value or quantile statistics	Distribution	No

Table 73. Column Distribution Statistics (SYSCAT.COLDIST and SYSSTAT.COLDIST) (continued)

Statistic	Description	RUNSTATS Option	
		Table	**Indexes**
SEQNO	Frequency ranking of a sequence number to help uniquely identify the row in the table	Distribution	No
COLVALUE	Data value for which frequency or quantile statistic is collected	Distribution	No
VALCOUNT	Frequency with which the data value occurs in column, or for quantiles, the number of values less than or equal to the data value (COLVALUE)	Distribution	No

Note:
1. Column distribution statistics are gathered by specifying the WITH DISTRIBUTION clause on the RUNSTATS command. Note that distribution statistics may **not** be gathered unless there is a sufficient lack of uniformity in the column values.
2. DISTCOUNT is collected only for columns that are the first key column in an index.

Related concepts:

- "Catalog statistics" on page 277
- "Statistics for modeling production databases" on page 304
- "Statistics for user-defined functions" on page 296

Distribution statistics

You can collect two kinds of data distribution statistics:

- Frequency statistics

  These statistics provide information about the column and the data value with the highest number of duplicates, the next highest number of duplicate values, and so on to the level specified by the value of the *num_freqvalues* database configuration parameter. To disable collection of frequent-value statistics, set *num_freqvalues* to 0.

  You can also set *num_freqvalues* as RUNSTATS options for each table or statistical view and for specific columns.

- Quantile statistics

  These statistics provide information about how data values are distributed in relation to other values. Called K-quantiles, these statistics represent the value V at or below which at least K values lie. You can compute a K-quantile by sorting the values in ascending order. The K-quantile value is the value in the Kth position from the low end of the range.

  To specify the number of sections into which the column data values should be grouped, set the *num_quantiles* database configuration parameter to a value between 2 and 32,767. The default value is 20, which ensures an optimizer estimation error of a maximum of plus or minus 2.5% for any equality or less-than or greater-than predicate and a maximum error of plus or minus 5% for any BETWEEN predicate. To disable collection of quantile statistics, set num_quantiles to 0 or 1.

You can set *num_quantiles* for each table or statistical view, and for specific columns.

Note: If you specify larger *num_freqvalues* and *num_quantiles* values, more CPU resources and memory, as specified by the *stat_heap_sz* database configuration parameter, are required when you execute RUNSTATS.

When to collect distribution statistics To decide whether distribution statistics should be created and updated for a given table or statistical view, consider the following two factors:

- Whether applications use static or dynamic SQL and XQuery statements.

  Distribution statistics are most useful for dynamic queries and static queries that do not use host variables. When using queries with host variables, the optimizer makes limited use of distribution statistics.
- Whether data in columns is distributed uniformly.

  Create distribution statistics if at least one column in the table has a highly "non-uniform" distribution of data and the column appears frequently in equality or range predicates; that is, in clauses such as the following:

```
WHERE C1 = KEY;
WHERE C1 IN (KEY1, KEY2, KEY3);
WHERE (C1 = KEY1) OR (C1 = KEY2) OR (C1 = KEY3);
WHERE C1 <= KEY;
WHERE C1 BETWEEN KEY1 AND KEY2;
```

Two types of non-uniform data distributions might occur, possibly together:

- Data might be clustered in one or more sub-intervals instead of being evenly spread out between the highest and lowest data value. Consider the following column, in which the data is clustered in the range (5,10):

C1
0.0
5.1
6.3
7.1
8.2
8.4
8.5
9.1
93.6
100.0

  Quantile statistics help the optimizer deal with this kind of data distribution.

  To help determine whether column data is not uniformly distributed, execute a query such as the following example:

```
SELECT C1, COUNT(*) AS OCCURRENCES
  FROM T1
GROUP BY C1
ORDER BY OCCURRENCES DESC;
```

- Duplicate data values might occur often. Consider a column in which data is distributed with the following frequencies:

Data Value	Frequency
20	5
30	10
40	10

Data Value	Frequency
50	25
60	25
70	20
80	5

To help the optimizer deal with duplicate values, create both quantile and frequent-value statistics.

When to collect index statistics only

You might collect statistics based only on index data in the following situations:
- A new index has been created since the RUNSTATS utility was run and you do not want to collect statistics again on the table data.
- There have been many changes to the data that affect the first column of an index.

What level of statistical precision to specify

To determine the precision with which distribution statistics are stored, you specify the database configuration parameters, *num_quantiles* and *num_freqvalues*. You can also specify these parameters as RUNSTATS options when you collect statistics for a table or for columns. The higher you set these values, the greater precision RUNSTATS uses when it create and updates distribution statistics. However, greater precision requires greater use of resources, both during RUNSTATS execution and in the storage required in the catalog tables.

For most databases, specify between 10 and 100 for the *num_freqvalues* database configuration parameter. Ideally, frequent-value statistics should be created such that the frequencies of the remaining values are either approximately equal to each other or negligible compared to the frequencies of the most frequent values. The database manager might collect less than this number, because these statistics will only be collected for data values that occur more than once. If you need to collect only quantile statistics, set *num_freqvalues* to zero.

To set the number of quantiles, specify between 20 and 50 as the setting of the *num_quantiles* database configuration parameter. A rough rule of thumb for determining the number of quantiles is:
- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P
- The number of quantiles should be approximately 100/P if the predicate is a BETWEEN predicate, and 50/P if the predicate is any other type of range predicate (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ″>″ predicates. In general, specify at least 10 quantiles. More than 50 quantiles should be necessary only for extremely non-uniform data. If you need only frequent value statistics, set *num_quantiles* to zero. If you set this parameter to “1”, because the entire range of values fits in one quantile, no quantile statistics are collected.

Related concepts:
- “Catalog statistics” on page 277
- “Catalog statistics tables” on page 286
- “Extended examples of distribution-statistics use” on page 299

- "Optimizer use of distribution statistics" on page 298

Related tasks:
- "Collecting catalog statistics" on page 279
- "Collecting distribution statistics for specific columns" on page 281

Related reference:
- "num_freqvalues - Number of frequent values retained " on page 479
- "num_quantiles - Number of quantiles for columns " on page 480

Detailed index statistics

If you execute RUNSTATS for indexes with the DETAILED clause, you collect statistical information about indexes that allows the optimizer to estimate how many data page fetches will be required, based on various buffer-pool sizes. This additional information helps the optimizer make better estimates of the cost of accessing a table through an index.

Note: When you collect detailed index statistics, RUNSTATS takes longer and requires more memory and CPU processing. The SAMPLED DETAILED option, for which information calculated only for a statistically significant number of entries, requires 2MB of the statistics heap. Allocate an additional 488 4K pages to the *stat_heap_sz* database configuration parameter setting for this additional memory requirement. If the heap appears to be too small, RUNSTATS returns an error before attempting to collect statistics.

The DETAILED statistics PAGE_FETCH_PAIRS and CLUSTERFACTOR will be collected only if the table is of a sufficient size: around 25 pages. In this case, CLUSTERFACTOR will be a value between 0 and 1; and CLUSTERRATIO will be -1 (not collected). For tables smaller than 25 pages, CLUSTERFACTOR will be -1 (not collected), and CLUSTERRATIO will be a value between 0 and 100; even if the DETAILED clause is specified for an index on that table.

The DETAILED statistics provide concise information about the number of physical I/Os required to access the data pages of a table if a complete index scan is performed under different buffer sizes. As RUNSTATS scans the pages of the index, it models the different buffer sizes, and gathers estimates of how often a page fault occurs. For example, if only one buffer page is available, each new page referenced by the index results in a page fault. In a worse case, each row might reference a different page, resulting in at most the same number of I/Os as rows in the indexed table. At the other extreme, when the buffer is big enough to hold the entire table (subject to the maximum buffer size), then all table pages are read once. As a result, the number of physical I/Os is a monotone, non-increasing function of the buffer size.

The statistical information also provides finer estimates of the degree of clustering of the table rows to the index order. The less the table rows are clustered in relation to the index, the more I/Os are required to access table rows through the index. The optimizer considers both the buffer size and the degree of clustering when it estimates the cost of accessing a table through an index.

You should collect DETAILED index statistics when queries reference columns that are not included in the index. In addition, DETAILED index statistics should be used in the following circumstances:

- The table has multiple unclustered indexes with varying degrees of clustering
- The degree of clustering is non-uniform among the key values
- The values in the index are updated non-uniformly

It is difficult to evaluate these conditions without previous knowledge or without forcing an index scan under varying buffer sizes and then monitoring the physical I/Os that result. Probably the cheapest way to determine whether any of these situations occur is to collect the DETAILED statistics for an index, examine them, and retain them if the PAGE_FETCH_PAIRS that result are non-linear.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics tables" on page 286

Related tasks:

- "Collecting catalog statistics" on page 279
- "Collecting index statistics" on page 282

Sub-element statistics

If tables contain columns that contain sub-fields or sub-elements separated by blanks, and queries reference these columns in WHERE clauses, you should collect sub-element statistics to ensure the best access plans.

For example, suppose a database contains a table, DOCUMENTS, in which each row describes a document, and suppose that in DOCUMENTS there is a column called KEYWORDS that contains a list of relevant keywords relating to this document for text retrieval purposes. The values in KEYWORDS might be as follows:

```
'database simulation analytical business intelligence'
'simulation model fruit fly reproduction temperature'
'forestry spruce soil erosion rainfall'
'forest temperature soil precipitation fire'
```

In this example, each column value consists of 5 sub-elements, each of which is a word (the keyword), separated from the others by one blank.

For queries that specify LIKE predicates on such columns using the % match_all character:

```
SELECT .... FROM DOCUMENTS WHERE KEYWORDS LIKE '%simulation%'
```

it is often beneficial for the optimizer to know some basic statistics about the sub-element structure of the column.

The following statistics are collected when you execute RUNSTATS with the LIKE STATISTICS clause:

SUB_COUNT
The average number of sub-elements.

SUB_DELIM_LENGTH
The average length of each delimiter separating each sub-element, where a delimiter, in this context, is one or more consecutive blank characters.

In the KEYWORDS column example, SUB_COUNT is 5, and SUB_DELIM_LENGTH is 1, because each delimiter is a single blank character.

The DB2_LIKE_VARCHAR registry variable affects the way in which the optimizer deals with a predicate of the form:

```
COLUMN LIKE '%xxxxxx'
```

where *xxxxxx* is any string of characters; that is, any LIKE predicate whose search value starts with a % character. (It might or might not end with a % character). These are referred to as ″wildcard LIKE predicates″. For all predicates, the optimizer has to estimate how many rows match the predicate. For wildcard LIKE predicates, the optimizer assumes that the COLUMN being matched contains a series of elements concatenated together, and it estimates the length of each element based on the length of the string, excluding leading and trailing % characters.

To examine the values of the sub-element statistics, query SYSIBM.SYSCOLUMNS. For example:

```
select substr(NAME,1,16), SUB_COUNT, SUB_DELIM_LENGTH
   from sysibm.syscolumns where tbname = 'DOCUMENTS'
```

Note: RUNSTATS might take longer if you use the LIKE STATISTICS clause. For example, RUNSTATS might take between 15% and 40%, and longer on a table with five character columns, if the DETAILED and DISTRIBUTION options are not used. If either the DETAILED or the DISTRIBUTION option is specified, the overhead percentage is less, even though the absolute amount of overhead is the same. If you are considering using this option, you should assess this overhead against improvements in query performance.

Related concepts:

- “Catalog statistics” on page 277
- “Catalog statistics tables” on page 286

Related tasks:

- “Collecting catalog statistics” on page 279
- “Collecting distribution statistics for specific columns” on page 281

Statistics for user-defined functions

To create statistical information for user-defined functions (UDFs), you edit the SYSSTAT.FUNCTIONS catalog view. If UDF statistics are available, the optimizer can use them when it estimates costs for various access plans. The RUNSTATS utility does not collect statistics for UDFs. If statistics are not available the statistic column values are -1 and the optimizer uses default values that assume a simple UDF.

The following table provides information about the statistic columns for which you can provide estimates to improve performance:

Table 74. Function Statistics (SYSCAT.FUNCTIONS and SYSSTAT.FUNCTIONS)

Statistic	Description
IOS_PER_INVOC	Estimated number of read/write requests executed each time a function is executed.

Table 74. Function Statistics (SYSCAT.FUNCTIONS and SYSSTAT.FUNCTIONS) (continued)

Statistic	Description
INSTS_PER_INVOC	Estimated number of machine instructions executed each time a function is executed.
IOS_PER_ARGBYTE	Estimated number of read/write requests executed per input argument byte.
INSTS_PER_ARGBYTES	Estimated number of machine instructions executed per input argument byte.
PERCENT_ARGBYTES	Estimated average percent of input argument bytes that the function will actually process.
INITIAL_IOS	Estimated number of read/write requests executed only the first/last time the function is invoked.
INITIAL_INSTS	Estimated number of machine instructions executed only the first/last time the function is invoked.
CARDINALITY	Estimated number of rows generated by a table function.

For example, consider a UDF (EU_SHOE) that converts an American shoe size to the equivalent European shoe size. (These two shoe sizes could be UDTs.) For this UDF, you might set the statistic columns as follows:

- INSTS_PER_INVOC: set to the estimated number of machine instructions required to:
  - Invoke EU_SHOE
  - Initialize the output string
  - Return the result.
- INSTS_PER_ARGBYTE: set to the estimated number of machine instructions required to convert the input string into a European shoe size.
- PERCENT_ARGBYTES: set to 100 indicating that the entire input string is to be converted
- INITIAL_INSTS, IOS_PER_INVOC, IOS_PER_ARGBYTE, and INITIAL_IOS: set each to 0, since this UDF only performs computations.

PERCENT_ARGBYTES would be used by a function that does not always process the entire input string. For example, consider a UDF (LOCATE) that accepts two arguments as input and returns the starting position of the first occurrence of the first argument within the second argument. Assume that the length of the first argument is small enough to be insignificant relative to the second argument and, on average, 75 percent of the second argument is searched. Based on this information, PERCENT_ARGBYTES should be set to 75. The above estimate of the average of 75 percent is based on the following additional assumptions:

- Half the time the first argument is not found, which results in searching the entire second argument.
- The first argument is equally likely to appear anywhere within the second argument, which results in searching half of the second argument (on average) when the first argument is found.

You can use INITIAL_INSTS or INITIAL_IOS to record the estimated number of machine instructions or read/write requests that are performed only the first or last time the function is invoked, such as to record the cost of setting up a scratchpad area.

To obtain information about I/Os and instructions used by a user-defined function, you can use output provided by your programming language compiler or by monitoring tools available for your operating system.

Related concepts:

- "Catalog statistics tables" on page 286
- "General rules for updating catalog statistics manually" on page 306

Optimizer use of distribution statistics

The optimizer uses distribution statistics for better estimates of the cost of various possible access plans to satisfy queries.

If you do not execute RUNSTATS with the WITH DISTRIBUTION clause, the catalog statistics tables contain information only about the size of the table or statistical view and the highest and lowest values in the table or statistical view, the degree of clustering of the table to any of its indexes, and the number of distinct values in indexed columns.

Unless it has additional information about the distribution of values between the low and high values, the optimizer assumes that data values are evenly distributed. If data values differ widely from each other, are clustered in some parts of the range, or contain many duplicate values, the optimizer will choose a less than optimal access plan.

Consider the following example:

The optimizer needs to estimate the number of rows containing a column value that satisfies an equality or range predicate in order to select the least expensive access plan. The more accurate the estimate, the greater the likelihood that the optimizer will choose the optimal access plan. For example, consider the query

```
SELECT C1, C2
  FROM TABLE1
  WHERE C1 = 'NEW YORK'
  AND C2 <= 10
```

Assume that there is an index on both C1 and C2. One possible access plan is to use the index on C1 to retrieve all rows with `C1 = 'NEW YORK'` and then check each retrieved row to see if `C2 <= 10`. An alternate plan is to use the index on C2 to retrieve all rows with `C2 <= 10` and then check each retrieved row to see if `C1 = 'NEW YORK'`. Because the primary cost in executing the query is usually the cost of retrieving the rows, the best plan is the plan that requires the fewest retrievals. Choosing this plan requires estimating the number of rows that satisfy each predicate.

When distribution statistics are not available but RUNSTATS has been executed against a table or a statistical view, the only information available to the optimizer is the second-highest data value (HIGH2KEY), second-lowest data value (LOW2KEY), number of distinct values (COLCARD), and number of rows (CARD) for a column. The number of rows that satisfy an equality or range predicate is

then estimated under the assumption that the frequencies of the data values in a column are all equal and the data values are evenly spread out over the interval (LOW2KEY, HIGH2KEY). Specifically, the number of rows satisfying an equality predicate C1 = KEY is estimated as CARD/COLCARD, and the number of rows satisfying a range predicate C1 BETWEEN KEY1 AND KEY2 is estimated as:

```
      KEY2 - KEY1
------------------  x CARD       (1)
 HIGH2KEY - LOW2KEY
```

These estimates are accurate only when the true distribution of data values in a column is reasonably uniform. When distribution statistics are unavailable and either the frequencies of the data values differ widely from each other or the data values are clustered in a few sub-intervals of the interval (LOW_KEY,HIGH_KEY), the estimates can be off by orders of magnitude and the optimizer may choose a less than optimal access plan.

When distribution statistics are available, the errors described above can be greatly reduced by using frequent-value statistics to compute the number of rows that satisfy an equality predicate and using frequent-value statistics and quantiles to compute the number of rows that satisfy a range predicate.

Related concepts:

- "Catalog statistics" on page 277
- "Distribution statistics" on page 291
- "Extended examples of distribution-statistics use" on page 299

Related tasks:

- "Collecting distribution statistics for specific columns" on page 281

Extended examples of distribution-statistics use

To understand how the optimizer might use distribution statistics, consider first a query that contains an equality predicate of the form C1 = KEY.

Example for Frequent-Value Statistics

If frequent-value statistics are available, the optimizer can use these statistics to choose an appropriate access plan, as follows:

- If KEY is one of the N most frequent values, then the optimizer uses the frequency of KEY that is stored in the catalog.
- If KEY is not one of the N most frequent values, the optimizer estimates the number of rows that satisfy the predicate under the assumption that the (COLCARD - N) non-frequent values have a uniform distribution. That is, the number of rows is estimated as:

```
CARD - NUM_FREQ_ROWS
--------------------          (2)
   COLCARD - N
```

where CARD is the number of rows in the table, COLCARD is the cardinality of the column and NUM_FREQ_ROWS is the total number of rows with a value equal to one of the N most frequent values.

For example, consider a column (C1) for which the frequency of the data values is as follows:

Data Value	Frequency
1	2
2	3
3	40
4	4
5	1

If frequent-value statistics based on only the most frequent value (that is, N = 1) are available, for this column, the number of rows in the table is 50 and the column cardinality is 5. For the predicate `C1 = 3`, exactly 40 rows satisfy it. If the optimizer assumes that data is evenly distributed, it estimates the number of rows that satisfy the predicate as 50/5 = 10, with an error of -75%. If the optimizer can use frequent-value statistics, the number of rows is estimated as 40, with no error.

Consider another example in which 2 rows satisfy the predicate `C1 = 1`. Without frequent-value statistics, the number of rows that satisfy the predicate is estimated as 10, an error of 400%. You may use the following formula to calculate the estimation error (as a percentage):

```
estimated rows  - actual rows
-----------------------------  X 100
          actual rows
```

Using the frequent value statistics (N = 1), the optimizer will estimate the number of rows containing this value using the formula (2) given above, for example:

```
(50 - 40)
--------- = 3
 (5 - 1)
```

and the error is reduced by an order of magnitude as shown below:

```
 3 - 2
------- = 50%
   2
```

Example for Quantile Statistics

The following explanations of quantile statistics use the term “K-quantile”. The *K-quantile* for a column is the smallest data value, `V`, such that at least “K” rows have data values less than or equal to `V`. To computer a K-quantile, sort the rows in the column according to increasing data values; the K-quantile is the data value in the Kth row of the sorted column.

If quantile statistics are available, the optimizer can better estimate the number of rows that satisfy a range predicate, as illustrated by the following examples. Consider a column (`C`) that contains the following values:

C
0.0
5.1
6.3
7.1
8.2
8.4
8.5
9.1
93.6

C
100.0

and suppose that K-quantiles are available for K = 1, 4, 7, and 10, as follows:

K	K-quantile
1	0.0
4	7.1
7	8.5
10	100.0

First consider the predicate `C <= 8.5`. For the data given above, exactly 7 rows satisfy this predicate. Assuming a uniform data distribution and using formula (1) from above, with KEY1 replaced by LOW2KEY, the number of rows that satisfy the predicate is estimated as:

```
 8.5 - 5.1
---------- x 10 *= 0
93.6 - 5.1
```

where *= means “approximately equal to”. The error in this estimation is approximately -100%.

If quantile statistics are available, the optimizer estimates the number of rows that satisfy this same predicate (`C <= 8.5`) by locating 8.5 as the highest value in one of the quantiles and estimating the number of rows by using the corresponding value of K, which is 7. In this case, the error is reduced to 0.

Now consider the predicate `C <= 10`. Exactly 8 rows satisfy this predicate. If the optimizer must assume a uniform data distribution and use formula (1), the number of rows that satisfy the predicate is estimated as 1, an error of -87.5%.

Unlike the previous example, the value 10 is not one of the stored K-quantiles. However, the optimizer can use quantiles to estimate the number of rows that satisfy the predicate as `r_1 + r_2`, where `r_1` is the number of rows satisfying the predicate `C <= 8.5` and `r_2` is the number of rows satisfying the predicate `C > 8.5 AND C <= 10`. As in the above example, `r_1 = 7`. To estimate `r_2` the optimizer uses linear interpolation:

```
       10 - 8.5
r_2 *= ---------- x (number of rows with value > 8.5 and <= 100.0)
       100 - 8.5
       10 - 8.5
r_2 *= ---------- x (10 - 7)
       100 - 8.5
       1.5
r_2 *= ---- x (3)
       91.5
r_2 *= 0
```

The final estimate is r_1 + r_2 *= 7, and the error is only -12.5%.

Quantiles improves the accuracy of the estimates in the above examples because the real data values are ″clustered″ in the range 5 - 10, but the standard estimation formulas assume that the data values are spread out evenly between 0 and 100.

The use of quantiles also improves accuracy when there are significant differences in the frequencies of different data values. Consider a column having data values

with the following frequencies:

Data Value	Frequency
20	5
30	5
40	15
50	50
60	15
70	5
80	5

Suppose that K-quantiles are available for K = 5, 25, 75, 95, and 100:

K	K-quantile
5	20
25	40
75	50
95	70
100	80

Also suppose that frequent value statistics are available based on the 3 most frequent values.

Consider the predicate `C BETWEEN 20 AND 30`. From the distribution of the data values, you can see that exactly 10 rows satisfy this predicate. Assuming a uniform data distribution and using formula (1), the number of rows that satisfy the predicate is estimated as:

```
30 - 20
-------  x 100 = 25
70 - 30
```

which has an error of 150%.

Using frequent-value statistics and quantiles, the number of rows that satisfy the predicate is estimated as `r_1 + r_2`, where `r_1` is the number of rows that satisfy the predicate `(C = 20)` and `r_2` is the number of rows that satisfy the predicate `C > 20 AND C <= 30`. Using formula (2), `r_1` is estimated as:

```
100 - 80
-------- = 5
  7 - 3
```

Using linear interpolation, `r_2` is estimated as:

```
  30 - 20
  ------- x (# rows with value > 20 and <= 40)
  40 - 20
  30 - 20
= ------- x (25 - 5)
  40 - 20
= 10,
```

yielding a final estimate of 15 and reducing the error by a factor of 3.

Related concepts:

- "Catalog statistics" on page 277
- "Distribution statistics" on page 291

- "Optimizer use of distribution statistics" on page 298
- "Rules for updating distribution statistics manually" on page 308

Related tasks:

- "Collecting distribution statistics for specific columns" on page 281

Catalog statistics for modeling and what-if planning

You can change the statistical information in the system catalogs so that it does not reflect the actual state of tables and indexes but allows you to examine various possible changes to the database for planning purposes. The ability to update selected system catalog statistics allows you to:

- Model query performance on a development system using production system statistics
- Perform "what-if" query performance analysis.

Do not manually update statistics on a production system. If you do, the optimizer might not choose the best access plan for production queries that contain dynamic SQL or XQuery statements.

Requirements

You must have explicit DBADM authority for the database to modify statistics for tables and indexes and their components. That is, your user ID is recorded as having DBADM authority in the SYSCAT.DBAUTH table. Belonging to a DBADM group does not explicitly provide this authority. A DBADM can see statistics rows for all users, and can execute SQL UPDATE statements against the views defined in the SYSSTAT schema to update the values of these statistical columns.

A user without DBADM authority can see only those rows which contain statistics for objects over which they have CONTROL privilege. If you do not have DBADM authority, you can change statistics for individual database objects if you have the following privileges for each object:

- Explicit CONTROL privilege on tables. You can also update statistics for columns and indexes for these tables.
- Explicit CONTROL privilege on nicknames in a federated database system. You can also update statistics for columns and indexes for these nicknames. Note that the update only affects local metadata (data-source table statistics are not changed). These updates affect only the global access strategy generated by the DB2 optimizer.
- Ownership of user-defined functions (UDFs)

The following shows an example of updating the table statistics for the EMPLOYEE table:

```
UPDATE SYSSTAT.TABLES
SET  CARD   = 10000,
     NPAGES = 1000,
     FPAGES = 1000,
     OVERFLOW = 2
WHERE TABSCHEMA = 'userid'
  AND TABNAME   = 'EMPLOYEE'
```

You must be careful when manually updating catalog statistics. Arbitrary changes can seriously alter the performance of subsequent queries. Even in a

non-production database that you are using for testing or modeling, you can use any of the following methods to refresh updates you applied to these tables and bring the statistics to a consistent state:

- ROLLBACK the unit of work in which the changes have been made (assuming the unit of work has not been committed).
- Use the RUNSTATS utility to recalculate and refresh the catalog statistics.
- Update the catalog statistics to indicate that statistics have not been gathered. (For example, setting column NPAGES to -1 indicates that the number-of-pages statistic has not been collected.)
- Replace the catalog statistics with the data they contained before you made any changes. This method is possible only if you used the *db2look* tool to capture the statistics before you made any changes.

In some cases, the optimizer may determine that some particular statistical value or combination of values is not valid. It will use default values and issue a warning. Such circumstances are rare, however, since most of the validation is done when updating the statistics.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics tables" on page 286
- "Statistics for modeling production databases" on page 304
- "Statistics for user-defined functions" on page 296

Statistics for modeling production databases

Sometimes you may want your test system to contain a subset of your production system's data. However, access plans selected for such a test system are not necessarily the same as those that would be selected on the production system, unless the catalog statistics and the configuration parameters for the test system are updated to match those of the production system.

A productivity tool, *db2look*, can be run against the production database to generate the update statements required to make the catalog statistics of the test database match those in production. These update statements can be generated by using *db2look* in mimic mode (`-m` option). In this case, *db2look* will generate a command processor script containing all the statements required to mimic the catalog statistics of the production database. This can be useful when analyzing SQL or XQuery statements through Visual Explain in a test environment.

You can recreate database data objects, including tables, views, indexes, and other objects in a database, by extracting DDL statements with *db2look -e*. You can run the command processor script created from this command against another database to recreate the database. You can use `-e` option and the `-m` option together in a script that re-creates the database and sets the statistics.

After running the update statements produced by `db2look` against the test system, the test system can be used to validate the access plans to be generated in production. Since the optimizer uses the type and configuration of the table spaces to estimate I/O costs, the test system must have the same table space geometry or layout. That is, the same number of containers of the same type, either SMS or DMS.

The *db2look* tool is found under the *bin* subdirectory.

For more information on how to use this productivity tool, type the following on a command line:

```
db2look -h
```

The Control Center also provides an interface to the *db2look* utility called "Generate DDL". Using the Control Center allows the results file from the utility to be integrated into the Script Center. You can also schedule the *db2look* command from the Control Center. One difference when using the Control Center is that only single table analysis can be done as opposed to a maximum of thirty tables in a single call using the *db2look* command. You should also be aware that LaTex and Graphical outputs are not supported from the Control Center.

You can also run the `db2look` utility against an OS/390 or z/OS database. The `db2look` utility extracts the DDL and UPDATE statistics statements for OS/390 objects. This is very useful if you would like to extract OS/390 or z/OS objects and re-create them in DB2 Database for Linux®, UNIX, and Windows.

There are some differences between statistics for DB2 Database for Linux, UNIX, and Windows, and the OS/390 statistics. The `db2look` utility performs the appropriate conversions from DB2 for OS/390 or z/OS to DB2 Database for Linux, UNIX and Windows when this is applicable and sets to a default value (-1) the DB2 Database for Linux, UNIX and Windows statistics for which a DB2 for OS/390 counterpart does not exist. Here is how the `db2look` utility maps the DB2 for OS/390 or z/OS statistics to those of DB2 Database for Linux, UNIX and Windows. In the discussion below, "UDB_x" stands for a DB2 Database for Linux, UNIX and Windows statistics column; and, "S390_x" stands for a DB2 for OS/390 or z/OS statistics column.

1. Table Level Statistics.

   UDB_CARD = S390_CARDF
   UDB_NPAGES = S390_NPAGES

   There is no S390_FPAGES. However, DB2 for OS/390 or z/OS has another statistics called PCTPAGES which represents the percentage of active table space pages that contain rows of the table. So it is possible to calculate UDB_FPAGES based on S390_NPAGES and S390_PCTPAGES as follows:

   ```
   UDB_FPAGES=(S390_NPAGES * 100)/S390_PCTPAGES
   ```

   There is no S390_OVERFLOW to map to UDB_OVERFLOW. Therefore, the `db2look` utility just sets this to the default value:

   ```
   UDB_OVERFLOW=-1
   ```

2. Column Level Statistics.

   UDB_COLCARD = S390_COLCARDF
   UDB_HIGH2KEY = S390_HIGH2KEY
   UDB_LOW2KEY = S390_LOW2KEY

   There is no S390_AVGCOLLEN to map to UDB_AVGCOLLEN so the `db2look` utility just sets this to the default value:

   ```
   UDB_AVGCOLLEN=-1
   ```

3. Index Level Statistics.

   UDB_NLEAF = S390_NLEAF
   UDB_NLEVELS = S390_NLEVELS
   UDB_FIRSTKEYCARD= S390_FIRSTKEYCARD
   UDB_FULLKEYCARD = S390_FULLKEYCARD
   UDB_CLUSTERRATIO= S390_CLUSTERRATIO

The other statistics for which there are no OS/390 or z/OS counterparts are just set to the default. That is:

```
UDB_FIRST2KEYCARD = -1
UDB_FIRST3KEYCARD = -1
UDB_FIRST4KEYCARD  = -1
UDB_CLUSTERFACTOR = -1
UDB_SEQUENTIAL_PAGES = -1
UDB_DENSITY  = -1
```

4. Column Distribution Statistics.

   There are two types of statistics in DB2 for OS/390 or z/OS SYSIBM.SYSCOLUMNS. Type "F" for frequent values and type "C" for cardinality. Only entries of type "F" are applicable to DB2 Database for Linux, UNIX, and Windows and these are the ones that will be considered.

   ```
   UDB_COLVALUE = S390_COLVALUE
   UDB_VALCOUNT = S390_FrequencyF * S390_CARD
   ```

   In addition, there is no column SEQNO in DB2 for OS/390 SYSIBM.SYSCOLUMNS. Because this required for DB2 for UDB, `db2look` generates one automatically.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "General rules for updating catalog statistics manually" on page 306

Related reference:

- "db2look - DB2 statistics and DDL extraction tool command" in *Command Reference*

Updating catalog statistics manually

General rules for updating catalog statistics manually

When you update catalog statistics, the most important general rule is to ensure that valid values, ranges, and formats of the various statistics are stored in the views for the statistics. It is also important to preserve the consistency of relationships between various statistics.

For example, COLCARD in SYSSTAT.COLUMNS must be less than CARD in SYSSTAT.TABLES (the number of distinct values in a column cannot be greater than the number of rows). Assume that you want to reduce COLCARD from 100 to 25, and CARD from 200 to 50. If you update SYSCAT.TABLES first, you should get an error (since CARD would be less than COLCARD). The correct order is to update COLCARD in SYSCAT.COLUMNS first, then update CARD in SYSSTAT.TABLES. The situation occurs in reverse if you want to increase COLCARD to 250 from 100, and CARD to 300 from 200. In this case, you must update CARD first, then COLCARD.

When a conflict is detected between an updated statistic and another statistic, an error is issued. However, errors may not always be issued when conflicts arise. In some situations, the conflict is difficult to detect and report in an error, especially if the two related statistics are in different catalogs. For this reason, you should be careful to avoid causing such conflicts.

The most common checks you should make, before updating a catalog statistic, are:

1. Numeric statistics must be -1 or greater than or equal to zero.
2. Numeric statistics representing percentages (for example, CLUSTERRATIO in SYSSTAT.INDEXES) must be between 0 and 100.

Note: For row types, the table level statistics NPAGES, FPAGES, and OVERFLOW are not updateable for a sub-table.

When a table is first created, system catalog statistics are set to -1 to indicate the table has no statistics. Until statistics are gathered, DB2 uses default values for SQL or XQuery statement compilation and optimization. Updating the table or index statistics may fail if the new values are inconsistent with the default values. Therefore it is recommended that you perform RUNSTATS after creating a table, before attempting to update the table or index statistics.

Related concepts:

- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "Rules for updating column statistics manually" on page 307
- "Rules for updating distribution statistics manually" on page 308
- "Rules for updating index statistics manually" on page 310
- "Rules for updating table and nickname statistics manually" on page 309
- "Statistics for modeling production databases" on page 304
- "Statistics for user-defined functions" on page 296

Rules for updating column statistics manually

When you are updating statistics in SYSSTAT.COLUMNS, follow the guidelines below.

- When manually updating HIGH2KEY and LOW2KEY in SYSSTAT.COLUMNS, follow the behavior of the generated values:
  - The values for HIGH2KEY, LOW2KEY must be valid values for the data type of the corresponding user column.
  - The length of HIGH2KEY, LOW2KEY values must be the smaller of 33 or the maximum length of the target column data type, not including additional quotes which can bring the length of the string up to 68. This means that only the first 33 characters of the value in the corresponding user column will be considered in determining the HIGH2KEY, LOW2KEY values.
  - The HIGH2KEY/LOW2KEY values are stored in such a way that they can be used on the SET clause of an UPDATE statement and without manipulation on cost calculations. For character strings, this means single quotes are added to the beginning and end of the string and an extra quote is added for every quote already in the string. Examples of user column values and their corresponding values in the HIGH2KEY,LOW2KEY are given in the table below.

Table 75. HIGH2KEY and LOW2KEY values for datatypes

Datatype in user column	User data	Corresponding HIGH2KEY, LOW2KEY value
INTEGER	–12	–12
CHAR	abc	'abc'

Table 75. HIGH2KEY and LOW2KEY values for datatypes (continued)

Datatype in user column	User data	Corresponding HIGH2KEY, LOW2KEY value
CHAR	ab'c	'ab''c'

  - HIGH2KEY should be greater than LOW2KEY whenever there are more than three distinct values in the corresponding column.
- The cardinality of a column (COLCARD statistic in SYSSTAT.COLUMNS) cannot be greater than the cardinality of its corresponding table or statistical view (CARD statistic in SYSSTAT.TABLES).
- The number of nulls in a column (NUMNULLS statistic in SYSSTAT.COLUMNS) cannot be greater than the cardinality of its corresponding table or statistical view (CARD statistic in SYSSTAT.TABLES).
- No statistics are supported for columns with data types: LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "General rules for updating catalog statistics manually" on page 306

Rules for updating distribution statistics manually

You update distribution statistics manually only to model a production database or perform what-if tests on an artificially constructed database. Do not update distribution statistics on a production database.

Make sure that all the statistics in the catalog are consistent. Specifically, for each column, the catalog entries for the frequent data statistics and quantiles must satisfy the following constraints:

- Frequent value statistics (in the SYSSTAT.COLDIST catalog). These constraints include:
  - The values in column VALCOUNT must be unchanging or decreasing for increasing values of SEQNO.
  - The number of values in column COLVALUE must be less than or equal to the number of distinct values in the column, which is stored in column COLCARD in catalog view SYSSTAT.COLUMNS.
  - The sum of the values in column VALCOUNT must be less than or equal to the number of rows in the column, which is stored in column CARD in catalog view SYSSTAT.TABLES.
  - In most cases, the values in the column COLVALUE should lie between the second-highest and second-lowest data values for the column, which are stored in columns HIGH2KEY and LOW2KEY, respectively, in catalog view SYSSTAT.COLUMNS. There may be one frequent value greater than HIGH2KEY and one frequent value less than LOW2KEY.
- Quantiles (in the SYSSTAT.COLDIST catalog). These constraints include:
  - The values in column COLVALUE must be unchanging or decreasing for increasing values of SEQNO
  - The values in column VALCOUNT must be increasing for increasing values of SEQNO
  - The largest value in column COLVALUE must have a corresponding entry in column VALCOUNT equal to the number of rows in the column

– In most cases, the values in the column COLVALUE should lie between the second-highest and second-lowest data values for the column, which are stored in columns HIGH2KEY and LOW2KEY, respectively, in catalog view SYSSTAT.COLUMNS.

Suppose that distribution statistics are available for a column C1 with "R" rows and you wish to modify the statistics to correspond to a column with the same relative proportions of data values, but with "(F x R)" rows. To scale up the frequent-value statistics by a factor of F, each entry in column VALCOUNT must be multiplied by F. Similarly, to scale up the quantiles by a factor of F, each entry in column VALCOUNT must be multiplied by F. If you do not follow these rules, the optimizer might use the wrong filter factor and cause unpredictable performance when you run the query.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "General rules for updating catalog statistics manually" on page 306

Rules for updating table and nickname statistics manually

The only statistical values that you can update in SYSTAT.TABLES are CARD, FPAGES, NPAGES, and OVERFLOW, and for MDC tables, ACTIVE_BLOCKS. Keep in mind that:

1. CARD must be greater than or equal to all COLCARD values in SYSSTAT.COLUMNS that correspond to that table.
2. CARD must be greater than NPAGES.
3. FPAGES must be greater than NPAGES.
4. NPAGES must be less than or equal to any ″Fetch″ value in the PAGE_FETCH_PAIRS column of any index (assuming this statistic is relevant for the index).
5. CARD must not be less than or equal to any ″Fetch″ value in the PAGE_FETCH_PAIRS column of any index (assuming this statistic is relevant for the index).

When working within a federated database system, use caution when manually providing or updating statistics on a nickname over a remote view. The statistical information, such as the number of rows this nickname will return, might not reflect the real cost to evaluate this remote view and thus might mislead the DB2 optimizer. Situations that can benefit from statistics updates include remote views defined on a single base table with no column functions applied on the SELECT list. Complex views may require a complex tuning process which might require that each query be tuned. Consider creating local views over nicknames instead so the DB2 optimizer knows how to derive the cost of the view more accurately.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "General rules for updating catalog statistics manually" on page 306

Rules for updating index statistics manually

When you update the statistics in SYSSTAT.INDEXES, follow the rules described below:

1. PAGE_FETCH_PAIRS (in SYSSTAT. INDEXES) must adhere to the following rules:
   - Individual values in the PAGE_FETCH_PAIRS statistic must be separated by a series of blank delimiters.
   - Individual values in the PAGE_FETCH_PAIRS statistic must not be longer than 10 digits and must be less than the maximum integer value (MAXINT = 2147483647).
   - There must always be a valid PAGE_FETCH_PAIRS value if the CLUSTERFACTOR is greater than zero.
   - There must be exactly 11 pairs in a single PAGE_FETCH_PAIR statistic.
   - Buffer size entries of PAGE_FETCH_PAIRS must be ascending in value.
   - Any buffer size value in a PAGE_FETCH_PAIRS entry cannot be greater than MIN(NPAGES, 524287) for 32-bit operating system or MIN(NPAGES, 2147483647) for 64-bit operating system where NPAGES is the number of pages in the corresponding table (in SYSSTAT.TABLES).
   - "Fetches" entries of PAGE_FETCH_PAIRS must be descending in value, with no individual "Fetches" entry being less than NPAGES. "Fetch" size values in a PAGE_FETCH_PAIRS entry cannot be greater than the CARD (cardinality) statistic of the corresponding table.
   - If buffer size value is the same in two consecutive pairs, then page fetch value must also be the same in both the pairs (in SYSSTAT.TABLES).

   A valid PAGE_FETCH_UPDATE is:

   ```
   PAGE_FETCH_PAIRS =
    '100 380 120 360 140 340 160 330 180 320 200 310 220 305 240 300
     260 300 280 300 300 300'
   ```

   where

   ```
   NPAGES = 300
   CARD   = 10000
   CLUSTERRATIO  =  -1
   CLUSTERFACTOR = 0.9
   ```

2. CLUSTERRATIO and CLUSTERFACTOR (in SYSSTAT.INDEXES) must adhere to the following rules:
   - Valid values for CLUSTERRATIO are -1 or between 0 and 100.
   - Valid values for CLUSTERFACTOR are -1 or between 0 and 1.
   - At least one of the CLUSTERRATIO and CLUSTERFACTOR values must be -1 at all times.
   - If CLUSTERFACTOR is a positive value, it must be accompanied by a valid PAGE_FETCH_PAIR statistic.
3. For relational indexes the following rules apply to FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, FULLKEYCARD, and INDCARD:
   - FIRSTKEYCARD must be equal to FULLKEYCARD for a single-column index.
   - FIRSTKEYCARD must be equal to COLCARD (in SYSSTAT.COLUMNS) for the corresponding column.
   - If any of these index statistics are not relevant, you should set them to -1. For example, if you have an index with only 3 columns, set FIRST4KEYCARD to -1.
   - For multiple column indexes, if all the statistics are relevant, the relationship between them must be:

```
FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
   <= FULLKEYCARD <= INDCARD == CARD
```

4. For indexes over XML data the following rules apply to FIRSTKEYCARD, FIRST2KEYCARD, FIRST3KEYCARD, FIRST4KEYCARD, FULLKEYCARD, and INDCARD:
   - The relationship between them must be:

     ```
     FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
        <= FULLKEYCARD <= INDCARD
     ```
5. The following rules apply to SEQUENTIAL_PAGES and DENSITY:
   - Valid values for SEQUENTIAL_PAGES are -1 or between 0 and NLEAF.
   - Valid values for DENSITY are -1 or between 0 and 100.

Related concepts:

- "Catalog statistics" on page 277
- "Catalog statistics for modeling and what-if planning" on page 303
- "Catalog statistics tables" on page 286
- "General rules for updating catalog statistics manually" on page 306

Automatic statistics collection

Automatic statistics collection

Automatic statistics collection

The DB2 optimizer uses catalog statistics to determine the most efficient access plan for any given query. Having out-of-date or incomplete statistics for a table or an index could lead the optimizer to select a plan that is not optimal, slowing down query execution. However, deciding which statistics to collect for a given workload is complex, and keeping these statistics up to date is time-consuming.

With automatic statistics collection, part of DB2's Automated Table Maintenance feature, you can let DB2 determine which statistics are required by your workload and which statistics need to be updated. With automatic statistics collection enabled, DB2 will automatically run the RUNSTATS utility in the background to ensure the correct statistics are collected and maintained.

Starting in DB2 Version 9, automatic statistics collection is enabled by default when a new database is created.

The performance impact of automatic statistics collection is minimized in several ways:

- Statistic collection is performed using throttled RUNSTATS. Throttling controls the amount of resources consumed by the RUNSTATS utility based on current database activity: as database activity increases, the RUNSTATS utility runs more slowly, reducing its resource demands.
- Only the minimal set of statistics for optimizing performance are collected. This is achieved through the use of statistics profiling which uses information about previous database activity to determine which statistics are required by the database workload, and how quickly those statistics will become out of date given the type of activity in the database.
- Only tables with high levels of activity (measured through the number of updates, deletes and inserts) are considered for statistic collection. Large tables (consisting of more than 4000 pages) are also sampled to determine whether the high table activity has indeed changed the statistics. Statistics for these large tables are only collected if warranted.

- The RUNSTATS utility is automatically scheduled to execute during the optimal maintenance window specified in your maintenance policy definition. This policy also specifies the set of tables that are within the scope of the automatic statistics collection, further minimizing unnecessary resource consumption.
- While automated statistic collection is being performed, the affected tables are still available for regular database activity (updates, inserts, deletes)as if RUNSTATS were not running on the table.

Related concepts:

- "Automatic features enabled by default" in *Administration Guide: Planning*
- "Collecting statistics using a statistics profile" on page 313

Related tasks:

- "Using automatic statistics collection" on page 312

Related reference:

- "RUNSTATS command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "auto_maint - Automatic maintenance " on page 481
- "util_impact_lim - Instance impact policy " on page 505
- "RUNSTATS command" in *Command Reference*

Using automatic statistics collection

Having accurate and complete database statistics is critical to efficient data access and optimal workload performance. Use the automatic statistics collection feature of the Automated Table Maintenance functionality to update and maintain relevant database statistics. You can optionally enhance this functionality in a serial environment by collecting query data and generating statistics profiles that help DB2 automatically collect the exact set of statistics required by your workload. This option is not available in MPP or federated environments, or when intra-partition parallelism is enabled.

Procedure:

You can turn this feature on using either the graphical user interface tools or the command line interface.

- To set up your database for automatic statistics collection using the graphical user interface tools:
  1. Open the Configure Automatic Maintenance wizard either from the Control Center by right-clicking on a database object or from the Health Center by right-clicking on the database instance that you want to configure for automatic statistics collection. Select **Configure Automatic Maintenance** from the pop-up window.
  2. Within this wizard, you can enable automatic statistics collection, specify the tables that you want to automatically collect statistics from, and specify a maintenance window for the execution of the RUNSTATS utility.
  3. OPTIONAL: To enable the automatic statistics profile generation, set the following two configuration parameters to "ON" using the command line interface:
     - AUTO_STATS_PROF
     - AUTO_PROF_UPD

- To set up your database for automatic statistics collection using the command line interface:
  1. Set each of the following configuration parameters to ″ON″:
     - AUTO_MAINT
     - AUTO_TBL_MAINT
     - AUTO_RUNSTATS
  2. OPTIONAL: To enable the automatic statistics profile generation, set the following two configuration parameters to ″ON″:
     - AUTO_STATS_PROF
     - AUTO_PROF_UPD

Related concepts:

- “Automatic statistics collection” on page 311
- “Collecting statistics using a statistics profile” on page 313

Related reference:

- “auto_maint - Automatic maintenance ” on page 481

Automatic statistics profiling

The RUNSTATS utility provides an option to register and use a statistics profile, which is a set of options that specify which statistics are to be collected on a particular table, for example, table statistics, index statistics, or distribution statistics.

Statistics profiles can now be generated automatically by the DB2 automatic statistics profiling feature. When this feature is enabled, information about database activity is collected and stored in a query feedback warehouse. Based on this data, a statistics profile is generated. Enabling this feature can alleviate the problem of uncertainty about which statistics are relevant to a particular workload and permits the collection of the minimal set of statistics to provide optimal database workload performance.

This feature can be used together with the automatic statistics collection feature to decide which statistics to collect on a particular table based on past table activity.

Related concepts:

- “Automating database maintenance” in *Deprecated Topics*
- “Automatic statistics profiling using automatic statistics collection” in *Administration Guide: Planning*
- “Collecting statistics using a statistics profile” on page 313
- “Guidelines for collecting and updating statistics” on page 283

Collecting statistics using a statistics profile

The RUNSTATS utility provides an option to register and use a statistics profile, which is a set of options that specify which statistics are to be collected on a particular table, for example, table statistics, index statistics, or distribution statistics.

This feature simplifies statistics collection by allowing you to store the options that you specify when you issue the RUNSTATS command so that you can collect the same statistics repeatedly on a table without having to re-type the command options.

You can register or update a statistics profile with or without actually collecting statistics. For example, to register a profile and collect statistics at the same time, issue the RUNSTATS command with the SET PROFILE option. To register a profile only, without actually collecting statistics, issue the RUNSTATS command with the SET PROFILE ONLY option.

To collect statistics using a statistics profile that you have already registered, issue the RUNSTATS command, specifying only the name of the table and the USE PROFILE option.

To see what options are currently specified in the statistics profile for a particular table, you can query the catalog tables with the following select statement, where `tablename` is the name of the table that you want the profile for:

```
SELECT STATISTICS_PROFILE FROM SYSIBM.SYSTABLES WHERE NAME = tablename
```

Automatic statistics profiling

Statistics profiles can also be generated automatically by the DB2 automatic statistics profiling feature. When this feature is enabled, information about database activity is collected and stored it in a query feedback warehouse. Based on this data, a statistics profile is generated. Enabling this feature can alleviate the problem of uncertainty about which statistics are relevant to a particular workload and permits the collection of the minimal set of statistics to provide optimal database workload performance.

This feature can be used with the automatic statistics collection feature, which automatically schedules statistics maintenance based on the information contained within the automatically generated statistics profile.

To enable this feature, you need to have already enabled automatic table maintenance by setting the appropriate configuration parameters . The AUTO_STATS_PROF configuration parameter activates the collection of query feedback data, and the AUTO_PROF_UPD configuration parameter activates the generation of a statistics profile for use by automatic statistics collection.

Note: Automatic statistics profile generation can only be activated in DB2 serial mode, and is blocked for queries in federated or multi-partition MPP environments, and where intra-partition parallelism is enabled.

Statistics profile generation is best suited to environments running large complex queries that apply many predicates, often having correlations in the data of the predicate columns, and joining and grouping over several tables. It is less suitable to environments with a primarily transactional workload.

There are a few different ways to use this feature:

- In a test environment. Set AUTO_STATS_PROF and AUTO_PROF_UPD to ON in test systems, where the performance overhead of runtime monitoring can be easily tolerated. When the test system uses realistic data and queries, this will allow for learning the proper correlations and settings of statistics parameters for RUNSTATS, which then will be stored in the statistics profiles. These profiles can then be transferred to the production system, where queries can benefit without incurring any monitoring overhead.

- To address performance issues for specific queries in a production environment. If performance problems for a particular set of queries is detected and can be attributed to faulty statistics or correlations, you can turn AUTO_STATS_PROF on and execute the target workload for a period of time. Automatic statistics profiling will analyze the query feedback and create recommendations in the SYSTOOLS.OPT_FEEDBACK_RANKING* tables. You can inspect these recommendations and refine the statistics profiles manually based on the recommendations. To have DB2 automatically update the statistics profiles based on these recommendations, turn AUTO_PROF_UPD on when you turn AUTO_STATS_PROF on.

Note: There is some performance overhead associated with monitoring the queries and storing the query feedback data in the feedback warehouse.

Creating the query feedback warehouse: The feedback warehouse consists of five tables in the SYSTOOLS schema that store information about the predicates encountered during query execution and recommendations for statistics collection. The five tables are OPT_FEEDBACK_QUERY, OPT_FEEDBACK_PREDICATE, OPT_FEEDBACK_PREDICATE_COLUMN, OPT_FEEDBACK_RANKING, and OPT_FEEDBACK_RANKING_COLUMN.

To use automatic statistics profiling, you need to first create the query feedback warehouse using the SYSINSTALLOBJECTS stored procedure. This stored procedure is the common stored procedure for creating and dropping objects in the SYSTOOLS schema.

Invoke the SYSINSTALLOBJECTS stored procedure as follows:

```
call SYSINSTALLOBJECTS ( toolname, action, tablespacename, schemaname)
```

where:

toolname
Specifies the name of the tool whose objects are to be created or dropped. In this case 'ASP' or 'AUTO STATS PROFILING'.

action Specifies the action to be taken: 'C' for create, 'D' for drop.

tablespacename
The name of the table space in which the the feedback warehouse tables will be created. This input parameter is optional. If it is not specified, the default user space will be used.

schemaname
The name of the schema with which the objects will be created or dropped. This parameter is currently not used.

For example, to create the feedback warehouse in table space "A" enter: `call SYSINSTALLOBJECTS ('ASP', 'C', 'A', '')`

Related concepts:

- "Automatic statistics collection" on page 311

Related tasks:

- "Using automatic statistics collection" on page 312

Chapter 13. Table and index maintenance

Table reorganization

After many changes to table data, logically sequential data may be on non-sequential physical data pages so that the database manager must perform additional read operations to access data. Additional read operations are also required if a significant number of rows have been deleted. In such a case, you might consider reorganizing the table to match the index and to reclaim space. You can reorganize the system catalog tables as well as database tables.

Note: Because reorganizing a table usually takes more time than running statistics, you might execute RUNSTATS to refresh the current statistics for your data and rebind your applications. If refreshed statistics do not improve performance, reorganization might help. For detailed information about the options and behavior of the REORG TABLE utility, refer to its command reference.

Consider the following factors, which might indicate that you should reorganize a table:

- A high volume of insert, update, and delete activity on tables accessed by queries
- Significant changes in the performance of queries that use an index with a high cluster ratio
- Executing RUNSTATS to refresh statistical information does not improve performance
- The REORGCHK command indicates a need to reorganize your table
- The tradeoff between the cost of increasing degradation of query performance and the cost of reorganizing your table, which includes the CPU time, the elapsed time, and the reduced concurrency resulting from the REORG utility locking the table until the reorganization is complete.

Reducing the need to reorganize tables

To reduce the need for reorganizing a table, perform these tasks after you create the table:

- Alter table to add PCTFREE
- Create clustering index with PCTFREE on index
- Sort the data
- Load the data

After you have performed these tasks, the table with its clustering index and the setting of PCTFREE on table helps preserve the original sorted order. If enough space is allowed in table pages, new data can be inserted on the correct pages to maintain the clustering characteristics of the index. As more data is inserted and the pages of the table become full, records are appended to the end of the table so that the table gradually becomes unclustered.

If you perform a REORG TABLE or a sort and LOAD after you create a clustering index, the index attempts to maintain a particular order of data, which improves the CLUSTERRATIO or CLUSTERFACTOR statistics collected by the RUNSTATS utility.

Note: Creating multi-dimensional clustering (MDC) tables might reduce the need to reorganize tables. For MDC tables, clustering is maintained on the columns that you specify as arguments to the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. However, REORGCHK might recommend reorganization of an MDC table if it considers that there are too many unused blocks or that blocks should be compacted.

Related concepts:

- "Snapshot monitor" in *System Monitor Guide and Reference*
- "DMS device considerations" in *Administration Guide: Planning*
- "DMS table spaces" in *Administration Guide: Planning*
- "Index reorganization" on page 318
- "SMS table spaces" in *Administration Guide: Planning*
- "Table and index management for MDC tables" on page 337
- "Table and index management for standard tables" on page 333

Related tasks:

- "Choosing a table reorganization method" on page 323
- "Determining when to reorganize tables" on page 320

Related reference:

- "REORG INDEXES/TABLE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "REORG INDEXES/TABLE command" in *Command Reference*

Index reorganization

As tables are updated with deletes and inserts, index performance degrades in the following ways:

- Fragmentation of leaf pages

  When leaf pages are fragmented, I/O costs increase because more leaf pages must be read to fetch table pages.
- The physical index page order no longer matches the sequence of keys on those pages, which is referred to as a *badly clustered* index.

  When leaf pages are badly clustered, sequential prefetching is inefficient and results in more I/O waits.
- The index develops more than its maximally efficient number of levels.

  In this case, the index should be reorganized.

If you set the MINPCTUSED parameter when you create an index, the database server automatically merges index leaf pages if a key is deleted and the free space is less than the specified percent. This process is called *online index defragmentation*. However, to restore index clustering, free space, and reduce leaf levels, you can use one of the following methods:

- Drop and recreate the index.
- Use the REORG INDEXES command to reorganize indexes online.

You might choose this method in a production environment because it allows users to read from and write to the table while its indexes are being rebuilt.

- Use the REORG TABLE command with options that allow you to reorganize both the table and its indexes off-line.

Online index reorganization

When you use the REORG INDEXES command with the ALLOW WRITE ACCESS option, all indexes on the specified table are rebuilt while read and write access to the table is allowed. Any changes made to the underlying table that would affect indexes while the reorganization is in progress are logged in the DB2 logs. In addition, the same changes are placed in the internal memory buffer space, if there is any such memory space available for use. The reorganization will process the logged changes to catch up with current writing activity while rebuilding the indexes. The internal memory buffer space is a designated memory area allocated on demand from the utility heap to store the changes to the index being created or reorganized. The use of the memory buffer space allows the index reorganization to process the changes by directly reading from memory first, and then reading through the logs if necessary, but at a much later time. The allocated memory is freed once the reorganization operation completes. Following the completion of the reorganization, the rebuilt index might not be perfectly clustered. If PCTFREE is specified for an index, that percent of space is preserved on each page during reorganization.

For partitioned tables, online index reorganization and cleanup of individual indexes is supported. For reorganization of individual indexes specify the index name: `REORG INDEX index_name for TABLE table_name`

Online index reorganization in ALLOW WRITE mode is not supported for spatial indexes or multi-dimensionally clustered (MDC) tables.

Note: The CLEANUP ONLY option of the REORG INDEXES command does not fully reorganize indexes. The CLEANUP ONLY ALL option removes keys that are marked deleted and are known to be committed. It also frees pages in which all keys are marked deleted and are known to be committed. When pages are freed, adjacent leaf pages are merged if doing so can leave at least PCTFREE free space on the merged page. PCTFREE is the percentage of free space defined for the index when it is created. The CLEANUP ONLY PAGES option deletes only pages in which all keys are marked deleted and are known to be committed.

When reorganizing indexes on partitioned tables using the CLEANUP ONLY option, any access level is supported. If the CLEANUP ONLY option is not specified, the default access level ALLOW NO ACCESS is the only supported access level.

REORG INDEXES has the following requirements:

- SYSADM, SYSMAINT, SYSCTRL or DBADM authority, or CONTROL privilege on the indexes and table
- An amount of free space in the table space where the indexes are stored equal to the current size of the index

  Consider placing indexes subject to reorganization in a large table space when you issue the CREATE TABLE statement.
- Additional log space

REORG INDEXES logs its activity. As a result, the reorganization might fail, especially if the system is busy and other concurrent activity is logged.

Note: If a REORG INDEXES ALL with the ALLOW NO ACCESS option fails, the indexes are marked bad and the operation is not undone. However, if a REORG with the ALLOW READ ACCESS or a REORG with the ALLOW WRITE ACCESS option fails, the original index object is restored.

Related concepts:

- "Using relational indexes to improve performance" on page 25
- "Relational index planning tips" on page 28
- "Online index defragmentation" on page 341
- "Relational index performance tips" on page 30
- "Index cleanup and maintenance" on page 340

Related tasks:

- "Choosing a table reorganization method" on page 323

Related reference:

- "REORG INDEXES/TABLE command" in *Command Reference*

Determining when to reorganize tables

Normal database activity, such as repeated inserts, deletes, and updates, can affect the organization of the data in your tables and indexes over time and adversely affect database performance. The RUNSTATS utility collects statistical information about the organization of the data in your database tables, allowing you to collect statistical information about table size and data distribution, the cluster ratio of indexes, the number of leaf pages in indexes, the number of table rows that overflow their original pages, and the number of filled and empty pages in a table. You can also use RUNSTATS to collect information about prefetch efficiency.

The information collected by RUNSTATS is stored in the system catalog tables. This information can help you decide when to reorganize tables and indexes and what type of reorganization is required.

In addtion to using RUNSTATS to capture information about your database organization at a single point in time, it is a good idea to run RUNSTATS regularly to identify overall trends that may be linked to changes in database performance.

The REORGCHK command also returns statistical information about data organization and can advise you about whether particular tables need to be reorganized. However, running specific queries against the catalog statistics tables at regular intervals or specific times can provide a performance history that allows you to spot trends that might have wider implications for performance.

Procedure:

To determine whether you need to reorganize tables, query the catalog statistics tables and monitor the following statistics:

1. Overflow of rows

   Query the OVERFLOW column in the SYSSTAT.TABLES table to monitor the overflow value. The values in this column represent the number of rows that

do not fit on their original pages. Row data can overflow when VARCHAR columns are updated with values that are longer than the initial values. In such cases, a pointer is kept at the original location in the row and the actual value is stored in another location that is indicated by the pointer. This can impact performance because the database manager must follow the pointer to find the contents of the row. This two-step process increases the processing time and might also increase the number of I/Os required.

Reorganizing the table data will eliminate the row overflows; therefore, as the number of overflow rows increases, the potential benefit of reorganizing your table data increases.

2. Fetch statistics

   Query the three following columns in the SYSCAT.INDEXES and SYSSTAT.INDEXES catalog statistics tables to determine the effectiveness of the prefetchers when the table is accessed in index order. These statistics characterize the average performance of the prefetchers against the underlying table.

   - The AVERAGE_SEQUENCE_FETCH_PAGES column stores the average number of pages that can be accessed in sequence in the table. Pages that can be accessed in sequence are eligible for prefetching. A small number indicates that the prefetchers are not as effective as they could be because they cannot read in the full number of pages specified by the PREFETCHSIZE setting for the table space. A large number indicates that the prefetchers are performing effectively. For a clustered index and table, this number should approach the value of NPAGES, the number of pages that contain rows.
   - The AVERAGE_RANDOM_FETCH_PAGES column stores the average number of random table pages between sequential page accesses when fetching table rows using the index. The prefetchers ignore small numbers of random pages when most pages are in sequence, and continue to prefetch to the configured prefetch size. As the table becomes more disorganized, the number of random fetch pages increases. Such disorganization is usually caused by inserts that occur out of sequence, either at the end of the table or in overflow pages. This causes fetches that slow query performance when the index is used to access a range of values.
   - The AVERAGE_SEQUENCE_FETCH_GAP column stores the average gap between table page sequences when fetching using the index. Detected through a scan of index leaf pages, each gap represents the average number of table pages that must be randomly fetched between sequences of table pages. These occur when many pages are accessed randomly, which interrupts the prefetchers. A large number indicates a table that is disorganized or poorly clustered to the index.

3. Number of index leaf pages that contain RIDs marked deleted but not removed

   In type-2 indexes, RIDs are not usually physically deleted when the RID is marked deleted. This means that useful space might be occupied by these logically deleted RIDs. To retrieve the number of leaf pages on which every RID is marked deleted, query the NUM_EMPTY_LEAFS column of the SYSCAT.INDEXES and SYSSTAT.INDEXES statistics tables. For leaf pages on which not all RIDs are marked deleted, the total number of logically deleted RIDs is stored in the NUMRIDS_DELETED column.

   Use this information to estimate how much space might be reclaimed by executing REORG INDEXES with the CLEANUP ALL option. To reclaim only the space in pages on which all RIDs are marked deleted, execute REORG INDEXES with the CLEANUP ONLY PAGES option.

4. Cluster-ratio and cluster-factor statistics for indexes

A cluster-ratio statistic is stored in the CLUSTERRATIO column of the SYSTCAT.INDEXES catalog table. This value, between 0 and 100, represents the degree of data clustering with the index. If you collect DETAILED index statistics, a finer cluster-factor statistic between 0 and 1 is stored in the CLUSTERFACTOR column instead, and the value of CLUSTERRATIO is –1. Only one of these two clustering statistics can be recorded in the SYSCAT.INDEXES catalog table. To compare CLUSTERFACTOR values with the CLUSTERRATIO values, multiply the CLUSTERFACTOR by 100 to obtain a percentage.

Note: In general, only one of the indexes in a table can have a high degree of clustering.

Index scans that are not index-only accesses might perform better with higher cluster ratios. A low cluster ratio leads to more I/O for this type of scan, since after the first access of each data page, it is less likely that the page is still in the buffer pool the next time it is accessed. Increasing the buffer size might also improve the performance of an unclustered index.

If table data was initially clustered on a certain index, and the clustering statistics information indicates that the data is now poorly clustered for that same index, you may want to reorganize the table to cluster the data again.

5. Number of leaf pages

   Query the NLEAF column in the SYSCAT.INDEXES table to find out the number of leaf pages occupied by an index. The number tells you how many index page I/Os are needed for a complete scan of an index.

   Ideally, an index should take up the minimum amount of space possible to reduce the I/Os required for an index scan. Random update activity can cause page splits that increase the size of the index.. When indexes are rebuilt during the reorganization of a table, each index can be built with the minimum amount of space.

   Note: By default, ten percent free space is left on each index page when the indexes are built. To increase the free space amount, specify the PCTFREE parameter when you create the index. Whenever you reorganize the index, the PCTFREE value is used. Free space greater than ten percent might reduce frequency of index reorganization because the additional space can accommodate additional index inserts.

6. Comparison of file pages

   To calculate the number of empty pages in a table, query the FPAGES and NPAGES columns in SYSCAT.TABLES and subtract the NPAGES number from the FPAGES number. The FPAGES column stores the total number of pages in use; the NPAGES column stores the number of pages that contain rows. Empty pages can occur when entire ranges of rows are deleted.

   As the number of empty pages increases, the need for a table reorganization increases. Reorganizing the table reclaims the empty pages and reduces the amount of space used by a table. In addition, because empty pages are read into the buffer pool for a table scan, reclaiming unused pages can improve the performance of a table scan.

   When the total number of pages (FPAGES) in a table is smaller than or equal to NPARTITIONS *1 extent size, table reorganization is not recommended. NPARTITIONS is the number of data partitions if this is a partitioned table, otherwise it is 1. In a DPF database, after the number of database partitions in a database partition group of the table is factored in, the condition changes to FPAGES ≤ the number of database partitions in a database partition group of a table * NPARTITIONS * 1 extent size

Related concepts:

- "Catalog statistics tables" on page 286
- "Index reorganization" on page 318
- "Table reorganization" on page 317

Related tasks:

- "Collecting catalog statistics" on page 279

Choosing a table reorganization method

DB2 provides two methods of reorganizing tables: classic and in-place. In general, classic table reorganization is faster, but should be used only if your applications function without write access to tables during the reorganization. If your environment does not allow this restriction, although in-place reorganization is slower, it can occur in the background while normal data access continues. Consider the features of each method and decide which method is more appropriate for your environment.

Procedure:

To choose a table reorganization method, consider the features of the following methods:

- **Classic table reorganization**

  This method provides the fastest table reorganization, especially if you do not need to reorganize LOB or LONG data. In addition, indexes are rebuilt in perfect order after the table is reorganized. Read-only applications can access the original copy of the table except during the last phases or the reorganization, in which the permanent table replaces the shadow copy of the table and the indexes are rebuilt.

  On the other hand, consider the following possible disadvantages:

  – Large space requirement

    Because classic table reorganization creates the shadow copy of the table, it can require twice as much space as the original table. If the reorganized table is larger than the original, reorganization can require more than twice as much space as the original.

    The shadow copy can be built in a temporary tablespace if the table tablespace is not large enough, but the replace phase performs best in the same DMS table space. Tables in SMS table spaces must always store the shadow copy in temporary space.

  – Limited table access

    Even read-only access is limited to the first phases of the reorganization process.

  – All or nothing process

    If the reorganization fails at any point, it must be restarted from the beginning on the nodes where it failed.

  – Performed within the controller of the application that invokes it

    The reorganization can be stopped only by that application or by a user who understands how to stop the process and has authority to execute the FORCE command for the application.

  Recommendation: Choose this method if you can reorganize tables during a maintenance window.

- **In-place table reorganization**

  The in-place method is slower and does not ensure perfectly ordered data, but it can allow applications to access the table during the reorganization. In addition, in-place table reorganization can be paused and resumed later by anyone with the appropriate authority by using the schema and table name.

  Note: In-place table reorganization is allowed only on tables with type-2 indexes and without extended indexes. In-place table reorganization is not allowed on tables with spatial indexes.

  Consider the following trade-offs:

  – Imperfect index reorganization

    You might need to reorganize indexes later to reduce index fragmentation and reclaim index object space.

  – Longer time to complete

    When required, in-place reorganization defers to concurrent applications. This means that long-running statements or RR and RS readers in long-running applications can slow the reorganization progress. In-place reorganization might be faster in an OLTP environment in which many small transactions occur.

  – Requires more log space

    Because in-place table reorganization logs its activities so that recovery is possible after an unexpected failure, it requires more log space than classic reorganization.

    It is possible that in-place reorganization will require log space equal to several times the size of the reorganized table. The amount of required space depends on the number of rows that are moved and the number and size of the indexes on the table.

  In-place table reorganization will reorganize table data objects only, and not indexes, long fields or LOBs.

  Recommendation: Choose in-place table reorganization for 24x7 operations with minimal maintenance windows.

Refer to the REORG TABLE syntax descriptions for detailed information about executing these table reorganization methods.

Monitoring the progress of table reorganization

Information about the current progress of table reorganization is written to the history file for database activity. The history file contains a record for each reorganization event. To view this file, execute the `db2 list history` command for the database that contains the table you are reorganizing.

You can also use table snapshots to monitor the progress of table reorganization. Table reorganization monitoring data is recorded regardless of the Database Monitor Table Switch setting.

If an error occurs, an SQLCA dump is written to the history file. For an in-place table reorganization, the status is recorded as PAUSED.

Related concepts:

- "Index reorganization" on page 318
- "Table reorganization" on page 317

Related tasks:

- "Determining when to reorganize tables" on page 320

Related reference:

- "REORG INDEXES/TABLE command" in *Command Reference*

Enabling automatic table and index reorganization

Having well-organized table data is critical to efficient data access and optimal workload performance. After many changes to table data, logically sequential data may be on non-sequential physical data pages so that the database manager must perform additional read operations to access data. Additional read operations are also required if a significant number of rows have been deleted. Use automated table reorganization to enable DB2 to manage offline table and index reorganization so that you don't have to worry about when and how to reorganize the data. You can enable DB2 to reorganize the system catalog tables as well as database tables.

Procedure:

You can turn this feature on using either the graphical user interface tools or the command line interface.

- To set up your database for automatic reorganization using the graphical user interface tools:
  1. Open the Configure Automatic Maintenance wizard either from the Control Center by right-clicking on a database object or from the Health Center by right-clicking on the database instance that you want to configure for automatic reorganization. Select **Configure Automatic Maintenance** from the pop-up window.
  2. Within this wizard, you can enable automatic reorganization to defragment data, specify the tables that you want to automatically reorganize, and specify a maintenance window for the execution of the REORG utility.
- To set up your database for automatic reorganization using the command line interface, set each of the following configuration parameters to ″ON″:
  - AUTO_MAINT
  - AUTO_TBL_MAINT
  - AUTO_REORG

Related concepts:

- "Automatic reorganization" in *Administration Guide: Planning*

Using snapshot monitor data to monitor the reorganization of a partitioned table

There is no separate data group indicating the overall table reorganization status for a partitioned table. A partitioned table uses a data organization scheme in which table data is divided across multiple storage objects, called data partitions or ranges, according to values in one or more table partitioning key columns of the table. However, you can deduce the global status of a table reorganization from the values of elements in the individual data partition data group being reorganized. The following information describes some of the most useful methods of monitoring the global status of a table reorganization.

Determining the number of data partitions being reorganized:

You can determine the total number of data partitions being reorganized on a table by counting the number of monitor data blocks for table data that have the same table name and schema name. This value indicates the number of data partitions on which reorganization has started. Examples 1 and 2 indicate that three data partitions are being reorganized.

Identifying the data partition being reorganized:

You can deduce the current data partition being reorganized from the phase start time (reorg_phase_start). During the SORT/BUILD/REPLACE phase, the monitor data corresponding to the data partition that is being reorganized shows the most recent phase start time. During the INDEX_RECREATE phase, the phase start time is the same for all the data partitions. In Examples 1 and 2, the INDEX_RECREATE phase is indicated, so the start time is the same for all the data partitions.

Identifying an index rebuild requirement:

You can determine if an index rebuild is required by obtaining the value of the maximum reorganize phase element (reorg_max_phase), corresponding to any one of the data partitions being reorganized. If reorg_max_phase has a value of 3 or 4, then an Index Rebuild is required. Examples 1 and 2 report a reorg_max_phase value of 3, indicating an index rebuild is required.

The following sample output is from a three-node server that contains a table with three data partitions:

```
CREATE TABLE  sales (c1 INT, c2 INT, c3 INT)
    PARTITION BY RANGE (c1)
        (PART P1 STARTING FROM (1)  ENDING AT (10) IN parttbs,
         PART P2 STARTING FROM (11) ENDING AT (20) IN parttbs,
         PART P3 STARTING FROM (21) ENDING AT (30) IN parttbs)
    DISTRIBUTE BY (c2)
```

Statement executed:

```
REORG TABLE sales ALLOW NO ACCESS ON ALL DBPARTITIONNUMS
```

Example 1:

```
GET SNAPSHOT FOR TABLES ON DPARTDB GLOBAL
```

The output is modified to include table information for the relevant table only.

```
           Table Snapshot

First database connect timestamp     = 06/28/2005 13:46:43.061690
Last reset timestamp                 = 06/28/2005 13:46:47.440046
Snapshot timestamp                   = 06/28/2005 13:46:50.964033
Database name                        = DPARTDB
Database path                        = /work/sales/NODE0000/SQL00001/
Input database alias                 = DPARTDB
Number of accessed tables            = 5

Table List
 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 0
 Data Object Pages   = 3
 Rows Read           = 12
```

```
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 0
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:49.816883
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.362918
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number       = 1
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:49.822701
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.420741
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.899543


Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:49.814813
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619
```

```
Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 1
Data Object Pages   = 3
Rows Read           = 8
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 0
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.014617
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.362918
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number       = 1
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.026278
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.420741
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.899543

Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.006392
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
```

```
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 2
Data Object Pages   = 3
Rows Read           = 4
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 0
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.199971
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.362918
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.821244

Table Reorg Information:
  Node number       = 1
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.223742
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.420741
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.899543

Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
```

```
  Reorg Tablespace  = 3
 Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.179922
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619
```

Example 2:

```
GET SNAPSHOT FOR TABLES ON DPARTDB AT DBPARTITIONNUM 2
```

The output is modified to include table information for the relevant table only.

```
           Table Snapshot

First database connect timestamp     = 06/28/2005 13:46:43.617833
Last reset timestamp                 =
Snapshot timestamp                   = 06/28/2005 13:46:51.016787
Database name                        = DPARTDB
Database path                        = /work/sales/NODE0000/SQL00001/
Input database alias                 = DPARTDB
Number of accessed tables            = 3

Table List
 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 0
 Data Object Pages   = 1
 Rows Read           = 0
 Rows Written        = 0
 Overflows           = 0
 Page Reorgs         = 0
 Table Reorg Information:
   Node number       = 2
   Reorg Type        =
        Reclaiming
        Table Reorg
        Allow No Access
        Recluster Via Table Scan
        Reorg Data Only
   Reorg Index       = 0
   Reorg Tablespace  = 3
 Long Temp space ID  = 3
   Start Time        = 06/28/2005 13:46:49.814813
   Reorg Phase       = 3 - Index Recreate
   Max Phase         = 3
   Phase Start Time  = 06/28/2005 13:46:50.344277
   Status            = Completed
   Current Counter   = 0
   Max Counter       = 0
   Completion        = 0
   End Time          = 06/28/2005 13:46:50.803619


 Table Schema        = NEWTON
 Table Name          = SALES
 Table Type          = User
 Data Partition Id   = 1
 Data Object Pages   = 1
 Rows Read           = 0
 Rows Written        = 0
```

```
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.006392
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619


Table Schema        = NEWTON
Table Name          = SALES
Table Type          = User
Data Partition Id   = 2
Data Object Pages   = 1
Rows Read           = 4
Rows Written        = 1
Overflows           = 0
Page Reorgs         = 0
Table Reorg Information:
  Node number       = 2
  Reorg Type        =
       Reclaiming
       Table Reorg
       Allow No Access
       Recluster Via Table Scan
       Reorg Data Only
  Reorg Index       = 0
  Reorg Tablespace  = 3
Long Temp space ID  = 3
  Start Time        = 06/28/2005 13:46:50.179922
  Reorg Phase       = 3 - Index Recreate
  Max Phase         = 3
  Phase Start Time  = 06/28/2005 13:46:50.344277
  Status            = Completed
  Current Counter   = 0
  Max Counter       = 0
  Completion        = 0
  End Time          = 06/28/2005 13:46:50.803619
```

Example 3:

```
SELECT * FROM SYSIBMADM.SNAPLOCK WHERE tabname = 'SALES';
```

The output is modified to include a subset of table information for the relevant table only.

```
...   TBSP_NAME TABNAME LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_STATUS ...
      --------- ------- ------------------ ---------- -----------
...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
...   -         SALES      TABLE_LOCK      IX         GRNT        ...
```

```
...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...
...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
...   -         SALES      TABLE_LOCK      IX         GRNT        ...
...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...
...   PARTTBS   SALES      ROW_LOCK        X          GRNT        ...
...   -         SALES      TABLE_LOCK      IX         GRNT        ...
...   PARTTBS   SALES      TABLE_PART_LOCK IX         GRNT        ...

 9 record(s) selected.
```

Output from this query (continued).

```
 ... LOCK_ESCALATION LOCK_ATTRIBUTES DATA_PARTITION_ID DBPARTITIONNUM
     --------------- --------------- ----------------- --------------
 ...               0 INSERT          2                 2
 ...               0 NONE            -                 2
 ...               0 NONE            2                 2
 ...               0 INSERT          0                 0
 ...               0 NONE            -                 0
 ...               0 NONE            0                 0
 ...               0 INSERT          1                 1
 ...               0 NONE            -                 1
 ...               0 NONE            1                 1
```

Example 4:

```
SELECT * FROM SYSIBMADM.SNAPTAB WHERE tabname = 'SALES';
```

The output is modified to include a subset of table information for the relevant table only.

```
... TABSCHEMA TABNAME TAB_FILE_ID TAB_TYPE   DATA_OBJECT_PAGES ROWS_WRITTEN ...
... --------- ------- ----------- ---------- ----------------- ------------ ...
... NEWTON    SALES             2 USER_TABLE                 1            1 ...
... NEWTON    SALES             4 USER_TABLE                 1            1 ...
... NEWTON    SALES             3 USER_TABLE                 1            1 ...

 3 record(s) selected.
```

Output from this query (continued).

```
... OVERFLOW_ACCESSES PAGE_REORGS DBPARTITIONNUM TBSP_ID DATA_PARTITION_ID
... ----------------- ----------- -------------- ------- -----------------
...                 0           0              0       3                 0
...                 0           0              2       3                 2
...                 0           0              1       3                 1
```

Example 5:

```
SELECT * FROM SYSIBMADM.SNAPTAB_REORG WHERE tabname = 'SALES';;
```

The output is modified to include a subset of table information for the relevant table only.

```
  REORG_PHASE   REORG_MAX_PHASE REORG_TYPE                                    ...
    ------------- --------------- -------------------------------------------
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
 INDEX_RECREATE               3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
```

```
INDEX_RECREATE             3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...
INDEX_RECREATE             3  RECLAIM+OFFLINE+ALLOW_NONE+TABLESCAN+DATAONLY  ...

 9 record(s) selected.
```

Output from this query (continued).

```
... REORG_STATUS REORG_TBSPC_ID DBPARTITIONNUM DATA_PARTITION_ID
    ------------------------ -------------- ----------------
... COMPLETED                3              2                0
... COMPLETED                3              2                1
... COMPLETED                3              2                2
... COMPLETED                3              1                0
... COMPLETED                3              1                1
... COMPLETED                3              1                2
... COMPLETED                3              0                0
... COMPLETED                3              0                1
... COMPLETED                3              0                2
```

Related concepts:

- "Data partitions" in *Administration Guide: Planning*
- "Partitioned tables" in *Administration Guide: Planning*
- "Snapshot monitor" in *System Monitor Guide and Reference*

Related reference:

- "data_partition_id - Data Partition Identifier monitor element" in *System Monitor Guide and Reference*
- "GET SNAPSHOT command" in *Command Reference*
- "reorg_max_phase - Maximum Reorganize Phase monitor element" in *System Monitor Guide and Reference*
- "reorg_phase_start - Reorganize Phase Start Time monitor element" in *System Monitor Guide and Reference*
- "Snapshot monitor interface mappings to logical data groups" in *System Monitor Guide and Reference*
- "Snapshot monitor logical data groups and monitor elements" in *System Monitor Guide and Reference*
- "Snapshot monitor sample output" in *System Monitor Guide and Reference*
- "SNAPTAB administrative view and SNAP_GET_TAB_V91 table function – Retrieve table logical data group snapshot information" in *Administrative SQL Routines and Views*
- "SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function – Retrieve table reorganization snapshot information" in *Administrative SQL Routines and Views*

Table and index management for standard tables

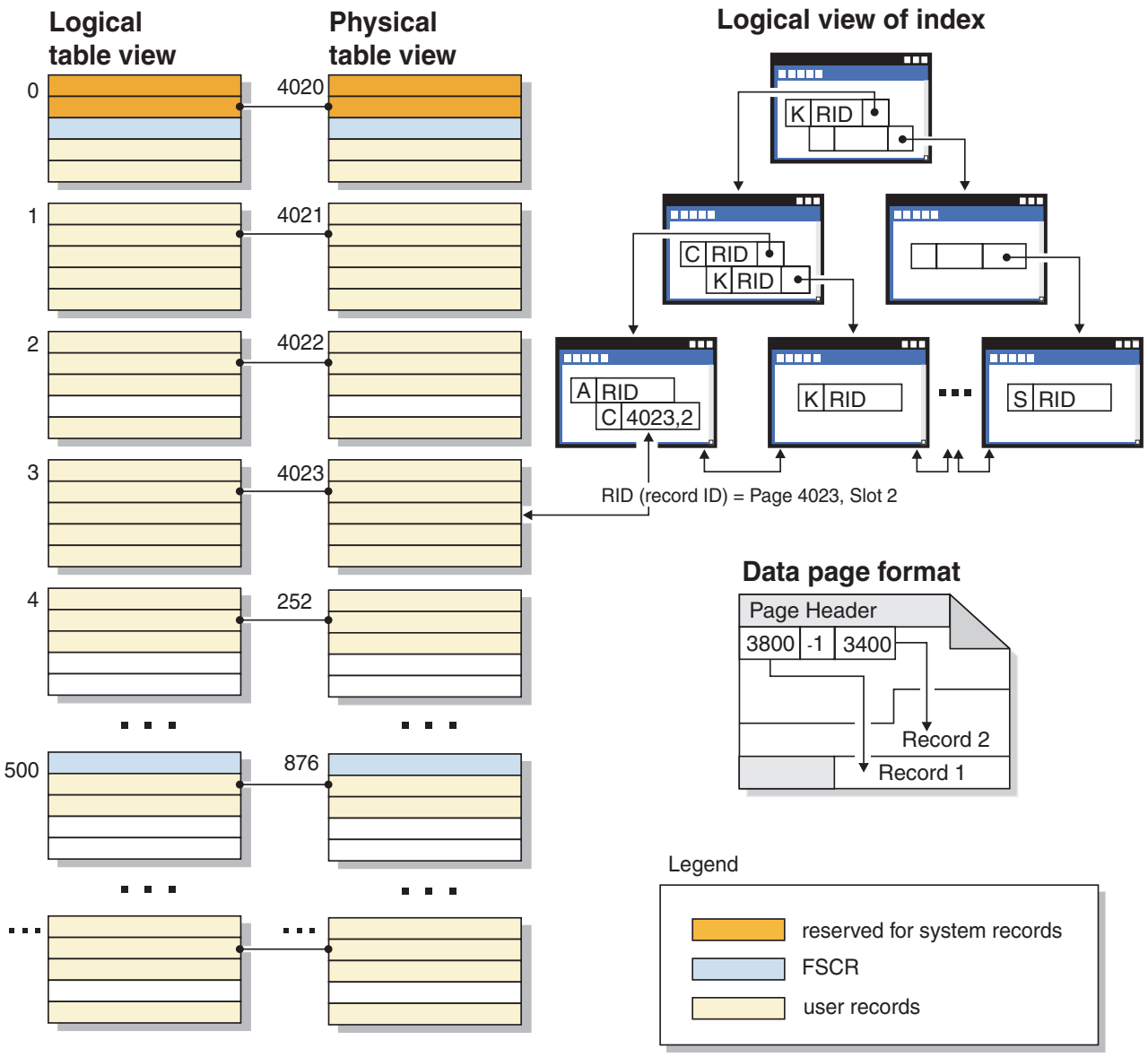


Figure 30. Logical table, record, and index structure for standard tables

In standard tables, data is logically organized as a list of data pages. These data pages are logically grouped together based on the extent size of the table space. For example, if the extent size is four, pages zero to three are part of the first extent, pages four to seven are part of the second extent, and so on.

The number of records contained within each data page can vary based on the size of the data page and the size of the records. A maximum of 255 records can fit on one page. Most pages contain only user records. However, a small number of pages include special internal records, that are used by DB2 to manage the table. For example, in a standard table there is a Free Space Control Record (FSCR) on every 500th data page. These records map the free space for new records on each of the following 500 data pages (until the next FSCR). This available free space is used when inserting records into the table.

Logically, index pages are organized as a B-tree which can efficiently locate records in the table that have a given key value. The number of entities on an index page

is not fixed but depends on the size of the key. For tables in DMS table spaces, record identifiers (RIDs) in the index pages use table space-relative page numbers, not object-relative page numbers. This allows an **index scan** to directly access the data pages without requiring an Extent Map page (EMP) for mapping.

Each data page has the same format. A page header begins each data page. After the page header there is a slot directory. Each entry in the slot directory corresponds to a different record on the page. The entry itself is the byte-offset into the data page where the record begins. Entries of minus one (-1) correspond to deleted records.

Record identifiers and pages

Record identifiers (RIDs) are a page number followed by a slot number. Type-2 index records also contain an additional field called the ridFlag. The ridFlag stores information about the status of keys in the index, such as whether this key has been marked deleted. Once the index is used to identify a RID, the RID is used to get to the correct data page and slot number on that page. Once a record is assigned a RID, it does not change until a table reorganization.

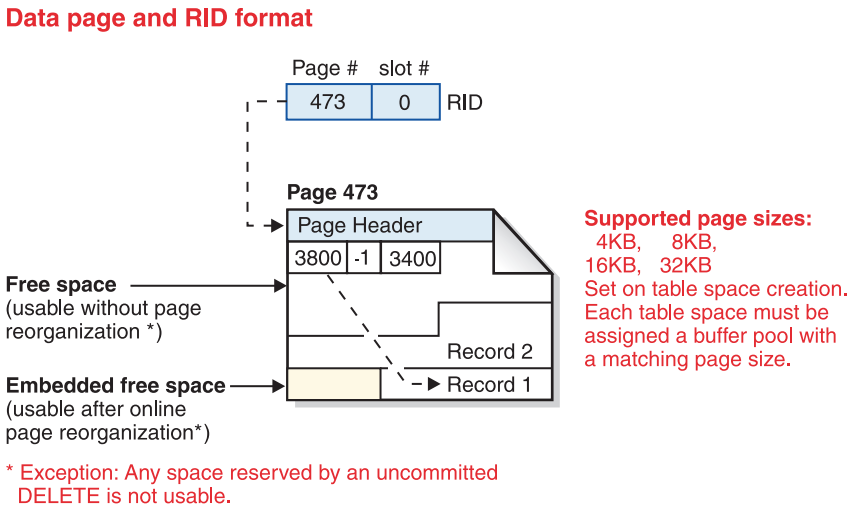


Figure 31. Data page and record-id (RID) format

When a table page is reorganized, embedded free space that is left on the page after a record is physically deleted is converted to usable free space. RIDs are redefined based on movement of records on a data page to take advantage of the usable free space.

DB2 supports different page sizes. Use larger page sizes for workloads that tend to access rows sequentially. For example, sequential access is used for Decision Support applications or where temporary tables are extensively used. Use smaller page sizes for workloads that tend to be more random in their access. For example, random access is used in OLTP environments.

Index management in standard tables

DB2 indexes use an optimized B-tree implementation based on an efficient and high concurrency index management method using write-ahead logging.

The optimized B-tree implementation has bi-directional pointers on the leaf pages that allows a single index to support scans in either forward or reverse direction. Index page are usually split in half except at the high-key page where a 90/10 split is used. That is, the high ten percent of the index keys are placed on a new page. This type of index page split is useful for workloads where INSERT requests are often completed with new high-keys.

Starting in Version 8.1, DB2 uses type-2 indexes. If you migrate from earlier versions of DB2, both type-1 and type-2 indexes are in use until you reorganize indexes or perform other actions that convert type-1 indexes to type-2. The index type determines how deleted keys are physically removed from the index pages.

- For type-1 indexes, keys are removed from the index pages during key deletion and index pages are freed when the last index key on the page is removed.
- For type-2 indexes, index keys are removed from the page during key deletion only if there is an X lock on the table. If keys cannot be removed immediately, they are marked deleted and physically removed later. For more information, refer to the section that describes type-2 indexes.

If you have enabled online index defragmentation by setting the MINPCTUSED clause to a value greater than zero when you created the index, index leaf pages can be merged online. The value that you specify is the threshold for the minimum percentage of space used on the index leaf pages. After a key is removed from an index page, if the percentage of space used on the page is at or below the value given, then the database manager attempts to merge the remaining keys with those of a neighboring page. If there is sufficient room, the merge is performed and an index leaf page is deleted. Online index defragmentation can improve space reuse, but if the MINPCTUSED value is too high then the time taken to attempt a merge increases and becomes less likely to succeed. The recommended value for this clause is fifty percent or less.

Note: Because online defragmentation occurs only when keys are removed from an index page, in a type-2 index it does not occur if keys are merely marked deleted, but have not been physically removed from the page.

The INCLUDE clause of the CREATE INDEX statement allows the inclusion of a specified column or columns on the index leaf pages in addition to the key columns. This can increase the number of queries that are eligible for index-only access. However, this can also increase the index space requirements and, possibly, index maintenance costs if the included columns are updated frequently. The maintenance cost of updating include columns is less than that of updating key columns, but more than that of updating columns that do not appear in the index. Ordering the index B-tree is only done using the key columns and not the included columns.

Related concepts:

- "Index cleanup and maintenance" on page 340
- "Improving insert performance" on page 128
- "Table and index management for MDC tables" on page 337
- "Designing multidimensional clustering (MDC) tables" in *Administration Guide: Planning*
- "Considerations when choosing table spaces for your tables" in *Administration Guide: Planning*
- "Multidimensional clustering (MDC) table creation, placement, and use" in *Administration Guide: Planning*

- "Space requirements for database objects" in *Administration Guide: Planning*

Related reference:

- "DB2 log records" in *Administrative API Reference*

Table and index management for MDC tables

Table and index organization for multi-dimensional clustering (MDC) tables is based on the same logical structures as standard table organization. Like standard tables, MDC tables are organized into pages that contain rows of data, divided into columns, and the rows on each page are identified by row IDs (RIDs). In addition, however, the pages of MDC tables are grouped into extent-sized blocks. For example, in the illustration below, which shows a table with an extent size of four, the first four pages, numbered 0 through 3, are the first block in the table. The next set of pages, numbered 4 through 7, are the second block in the table.

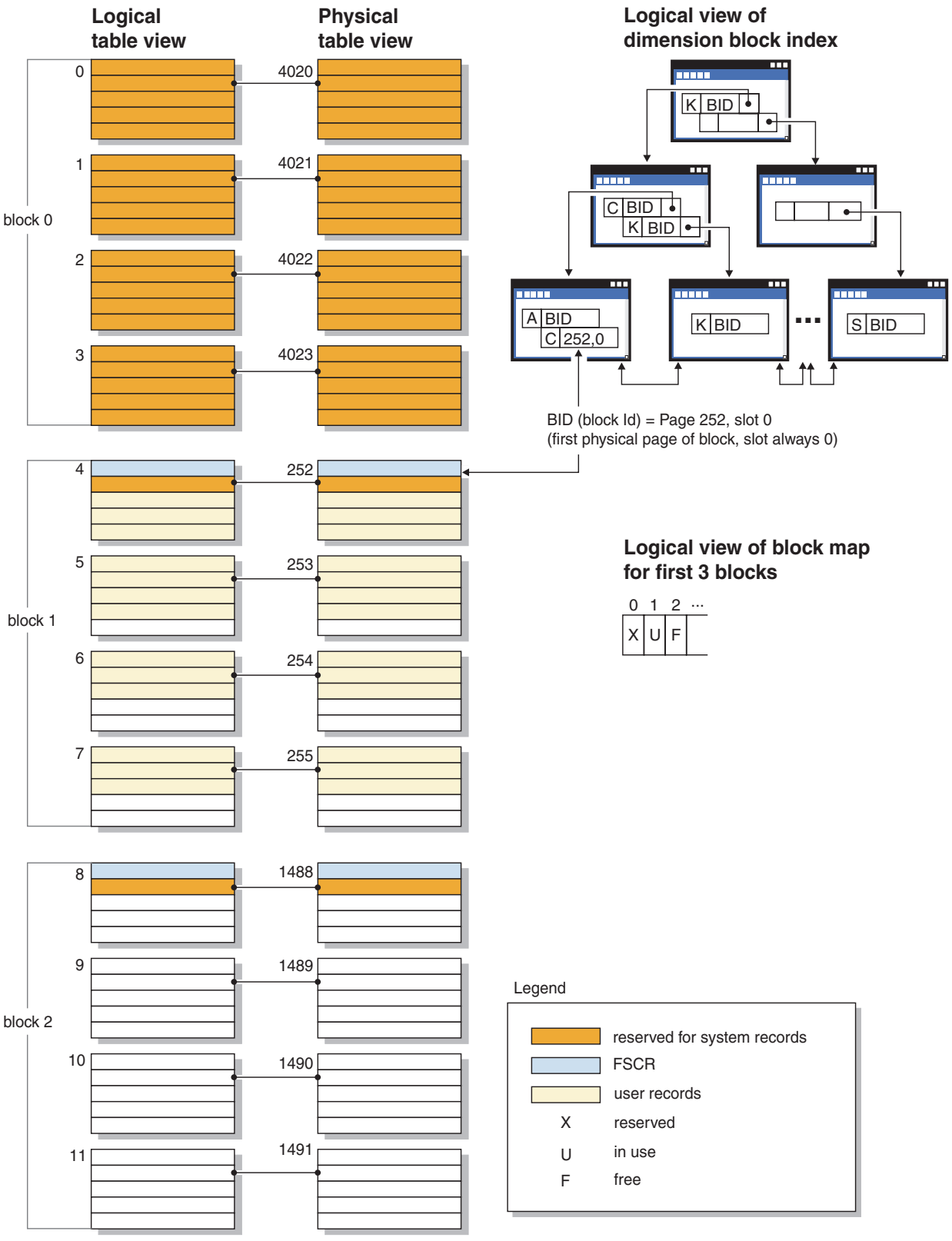


Figure 32. Logical table, record, and index structure for MDC tables

The first block contains special internal records that are used by DB2 to manage the table, including the free-space control record (FSCR). In subsequent blocks, the

first page contains the FSCR. An FSCR maps the free space for new records that exists on each of the pages in the block. This available free space is used when inserting records into the table.

As the name implies, MDC tables cluster data on more than one dimension. Each dimension is determined by a column or set of columns that you specify in the ORGANIZE BY DIMENSIONS clause of the CREATE TABLE statement. When you create an MDC table, the following two kinds of indexes are created automatically:

- A dimension-block index, which contains pointers to each occupied block for a single dimension.
- A composite block index, which contains all dimension key columns. The composite block index is used to maintain clustering during insert and update activity.

The optimizer considers access plans which utilize dimension-block indexes when it determines the most efficient access plan for a particular query. When queries have predicates on dimension values, the optimizer can use the dimension block index to identify, and fetch from, the extents that contain these values. Because extents are physically contiguous pages on disk, this results in more efficient performance and minimizes I/O.

In addition, you can create specific RID indexes if analysis of data access plans indicates that such indexes would improve query performance.

Along with the dimension block indexes and the composite block index, MDC tables maintain a block map that contains a bitmap that indicates the availability status of each block. The following attributes are coded in the bitmap list:

- X (reserved): the first block contains only system information for the table.
- U (in use): this block is used and associated with a dimension block index
- L (loaded): this block has been loaded by a current load operation
- C (check constraint): this block is set by the load operation to specify incremental constraint checking during the load.
- T (refresh table): this block is set by the load operation to specify that AST maintenance is required.
- F (free): If no other attribute is set, the block is considered free.

Because each block has an entry in the block map file, the file grows as the table grows. This file is stored as a separate object. In an SMS tablespace it is a new file type. In a DMS table space, it has a new object descriptor in the object table.

Related concepts:

- "Designing multidimensional clustering (MDC) tables" in *Administration Guide: Planning*
- "Multidimensional clustering (MDC) table creation, placement, and use" in *Administration Guide: Planning*
- "Space requirements for database objects" in *Administration Guide: Planning*
- "Improving insert performance" on page 128

Index cleanup and maintenance

After you create indexes, performance degrades unless you keep the index compact and organized. Consider the following suggestions to keep indexes as small and efficient as possible:

- Enable online index defragmentation

  Create indexes with the MINPCTUSED clause. Drop and recreate existing indexes, if necessary.
- Perform frequent COMMITs or get X locks on tables, either explicitly or by lock escalation, if frequent COMMITS are not possible.

  Index keys marked deleted can be physically removed from the table after the COMMIT. X locks on tables allow the deleted key to be physically removed when it is marked deleted, as explained below.
- Use REORGCHK to help determine when to reorganize indexes or tables, or both, and when to use the REORG INDEXES with the CLEANUP ONLY option.

  To allow read and write access to the index during reorganization, run REORG INDEXES with the ALLOW WRITE ACCESS option.

Note: In DB2 Version 8.1 and later, all new indexes are created as type-2 indexes. The one exception is when you add an index on a table that already has type-1 indexes. In this case only, the new index will also be a type-1 index. To find out what type of index exists for a table, execute the INSPECT command. To convert type-1 indexes to type-2 indexes, execute the REORG INDEXES command.

The primary advantages of type-2 indexes are as follows:

- An index can be created on columns whose length is greater than 255 bytes.
- The use of next-key locking is reduced to a minimum, which improves concurrency. Most next-key locking is eliminated because a key is marked deleted instead of being physically removed from the index page. For information about key locking, refer to topics that discuss the performance implications of locks.

Index keys that are marked deleted are cleaned up in the following circumstances:

- During subsequent insert, update, or delete activity

  During key insertion, keys that are marked deleted and are known to be committed are cleaned up if such a cleanup might avoid the need to perform a page split and prevent the index from increasing in size.

  During key deletion, when all keys on a page have been marked deleted an attempt is made to find another index page where all the keys are marked deleted and all those deletions have committed. If such a page is found, it is deleted from the index tree.

  If there is an X lock on the table when a key is deleted, the key is physically deleted instead of just being marked deleted. During this physical deletion, any deleted keys on the same page are also removed if they are marked deleted and known to be committed.
- When you execute the REORG INDEXES command with CLEANUP options

  The CLEANUP ONLY PAGES option searches for and frees index pages on which all keys are marked deleted and known to be committed.

The CLEANUP ONLY ALL option frees not only index pages on which all keys are marked deleted and known to be committed, but it also removes RIDs marked deleted and known to be committed on pages that contain some undeleted RIDs.

This option also tries to merge adjacent leaf pages if doing so results in a merged leaf page that has at least PCTFREE free space on the merged leaf page. The PCTFREE value is the percent of free space defined for the index when it is created. The default PCTFREE is ten percent. If two pages can be merged, one of the pages will be freed.

For partitioned tables, you are encouraged to execute RUNSTATS after an asynchronous index cleanup has completed in order to generate accurate index statistics in the presence of detached data partitions. To determine whether or not there are detached data partitions in the table, you can check the status field in the SYSDATAPARTITIONS table and look for the value ″I″ (index cleanup) or ″D″ (detached with dependant MQT).

- Any rebuild of an index

  Utilities that rebuild indexes include the following:

  – REORG INDEXES when not using one of the CLEANUP options
  – REORG TABLE when not using the INPLACE option
  – IMPORT with the REPLACE option
  – LOAD with the INDEXING MODE REBUILD option

Related concepts:

- “Using relational indexes to improve performance” on page 25
- “Relational index planning tips” on page 28
- “Index reorganization” on page 318
- “Index structure” on page 26
- “Table reorganization” on page 317

Online index defragmentation

Online index defragmentation is enabled by the user-definable threshold for the minimum amount of used space on an index leaf page. When an index key is deleted from a leaf page and the threshold is exceeded, the neighboring index leaf pages are checked to determine if two leaf pages can be merged. If there is sufficient space on a page for a merge of two neighboring pages to take place, the merge occurs immediately in the background.

Online defragmentation of the index is only possible with indexes created in Version 6 and later. If existing indexes require the ability to be merged online, they must be dropped and then re-created with the `MINPCTUSED` clause. Set the `MINPCTUSED` value to less than one hundred (100). The recommended value for `MINPCTUSED` is less than 50 because the goal is to merge two neighboring index leaf pages. A value of zero for `MINPCTUSED`, which is also the default, disables online defragmentation.

Pages in the index are freed when the last index key on the page is removed. The exception to this rule occurs when you specify `MINPCTUSED` clause in the CREATE INDEX statement. The `MINPCTUSED` clause specifies a percent of space on an index leaf page. When an index key is deleted, if the percent of filled space on the page is at or below the specified value, then the database manager tries to merge the

remaining keys with keys on an adjacent page. If there is sufficient space on an adjacent page, the merge is performed and an index leaf page is deleted.

Index non-leaf pages are not merged during an online index defragmentation. However, empty non-leaf pages are deleted and made available for re-use by other indexes on the same table. To free these non-leaf pages for other objects in a DMS storage model or to free disk space in an SMS storage model, perform a full reorganization of the table or indexes. Full reorganization of the table and indexes can make the index as small as possible. Index non-leaf pages are not merged during an online index defragmentation, but are deleted and freed for re-use if they become empty. The number of levels in the index and the number of leaf and non-leaf pages might be reduced.

For type-2 indexes, keys are removed from a page during key deletion only when there is an X lock on the table. During such an operation, online index defragmentation will be effective. However, if there is not an X lock on the table during key deletion, keys are marked deleted but are not physically removed from the index page. As a result, no defragmentation is attempted.

To defragment type-2 indexes in which keys are marked deleted but remain in the physical index page, execute the REORG INDEXES command with the CLEANUP ONLY ALL option. The CLEANUP ONLY ALL option defragments the index, regardless of the value of MINPCTUSED. If you execute REORG INDEXES with the CLEANUP ONLY ALL, two neighbouring leaf pages are merged if such a merge can leave at least PCTFREE free space on the merged page. PCTFREE is specified at index creation time and defaults to ten percent.

Related concepts:

- "Using relational indexes to improve performance" on page 25
- "Index reorganization" on page 318
- "Index structure" on page 26
- "Relational index performance tips" on page 30

Chapter 14. Data redistribution

Data redistribution

To redistribute table data among the database partitions in a partitioned database environment, you use the REDISTRIBUTE DATABASE PARTITION GROUP command.

Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

In a partitioned database you might redistribute data for the following reasons:

- To balance data volumes and processing loads across database partitions.

  Performance improves if data access can be spread out over more than one database partition.
- To introduce skew in the data distribution across database partitions.

  Access and throughput performance might improve if you redistribute data in a frequently accessed table so that infrequently accessed data is on a small number of database partitions in the database partition group, and the frequently accessed data is distributed over a larger number of database partitions. This would improve access performance and throughput on the most frequently run applications.

To preserve table collocation, use the REDISTRIBUTE DATABASE PARTITION GROUP command to redistribute data at the database partition group level. All tables are redistributed in a single operation. To achieve a specified data distribution, the REDISTRIBUTE DATABASE PARTITION GROUP command divides tables among the database database partitions as it moves the rows. Depending on the option you specify, the utility can either generate a target distribution map or use an existing distribution map as input.

How data is redistributed across database partitions

Data redistribution is performed on the set of tables in the specified database partition group of a database. You must connect to the database at the catalog database partition before executing REDISTRIBUTE DATABASE PARTITION GROUP command to invoke the Data Redistribution utility. The utility uses both the source distribution map and the target distribution map to identify which hash database partitions have been assigned to a new location, which is a new database partition number. All rows that correspond to a database partition that has a new location are moved from the database partition specified in the source distribution map to the database partition specified in the target distribution map.

The Data Redistribution utility performs the following steps:

1. Obtains a new distribution map ID for the target distribution map, and inserts it into the SYSCAT.PARTITIONMAPS catalog view.
2. Updates the REBALANCE_PMAP_ID column in the SYSCAT.DBPARTITIONGROUPS catalog view for the database partition group with the new distribution map ID.
3. Adds any new database partitions to the SYSCAT.DBPARTITIONGROUPDEF catalog view.

4. Sets the IN_USE column in the SYSCAT.DBPARTITIONGROUPDEF catalog view to 'D' for any database partition that is to be dropped.
5. Does a COMMIT for the catalog updates.
6. Creates database files for all new database partitions.
7. Redistributes the data on a table-by-table basis for every table in the database partition group, in the following steps:
   a. Locks the row for the table in the SYSTABLES catalog table.
   b. Invalidates all packages that involve this table. The distribution map ID associated with the table changes because the table rows are redistributed. Because the packages are invalidated, the compiler must obtain the new database partitioning information for the table and generate packages accordingly.
   c. Locks the table in exclusive mode.
   d. Uses DELETEs and INSERTs to redistribute the data in the table.
   e. If the redistribution operation succeeds, it issues a COMMIT for the table and continues with the next table in the database partition group. If the operation fails before the table is fully redistributed, the utility Issues a ROLLBACK on updates to the table, ends the entire redistribution operation and returns an error.
8. Deletes database files and deletes entries in the SYSCAT.NODEGROUPDEF catalog view for database partitions that were previously marked to be dropped.
9. Updates the database partition group record in the SYSCAT.NODEGROUPS catalog view to set PMAP_ID to the value of REBALANCE_PMAP_ID and REBALANCE_PMAP_ID to NULL.
10. Deletes the old distribution map from the SYSCAT.PARTITIONMAPS catalog view.
11. Does a COMMIT for all changes.

Related concepts:

- "Log space requirements for data redistribution" on page 347
- "Redistribution error recovery" on page 350

Related tasks:

- "Determining whether to redistribute data" on page 344
- "Redistributing data across database partitions" on page 345

Determining whether to redistribute data

Before you decide to redistribute data, find out whether data is distributed unequally among database partitions. After you have current distribution information, you can use this information to create a custom redistribution file or distribution map.

Procedure:

To get information about current data distributions for database partitions in a database partition group:

1. Determine if any database partitions have unequal distributions of rows.

   For the largest table, use an appropriate database partitioning column and enter a query such as the following:

```
SELECT PARTITION(column_name), COUNT(*) FROM table_name
   GROUP BY PARTITION(column_name)
   ORDER BY PARTITION(column_name) DESC
   FETCH FIRST 100 ROWS ONLY
```

The PARTITION and DBPARTITIONNUM SQL functions determine the current data distribution across hash partitions or database partitions. The PARTITION function returns the distribution map index for each row of the table. The DBPARTITIONNUM function returns the database partition number of the row.

2. Execute this query for other large tables that are distributed across the database partition group.
3. Use the information to create both a distribution file and a target distribution map.

Related concepts:

- "Data redistribution" on page 343
- "Log space requirements for data redistribution" on page 347

Related tasks:

- "Redistributing data across database partitions" on page 345

Redistributing data across database partitions

In a partitioned database environment, you might redistribute data among database partitions to balance data access in the following cases:

- When some database partitions contain more data than others
- When some database partitions are accessed more frequently than others

Prerequisites:

Log file size: Ensure that log files are large enough for the data redistribution operation. The log file on each affected database partition must be large enough to accommodate the INSERT and DELETE operations performed there.

Replicated materialized query tables: If the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.

Restrictions:

You can do the following operations on objects of the database partition group while the utility is running. You cannot, however, do them on the table that is being redistributed. You can:

- Create indexes on other tables. The CREATE INDEX statement uses the distribution map of the affected table.
- Drop other tables. The DROP TABLE statement uses the distribution map of the affected table.
- Drop indexes on other tables. The DROP INDEX statement uses the distribution map of the affected table.
- Query other tables.
- Update other tables.

- Create new tables in a table space defined in the database partition group. The CREATE TABLE statement uses the target distribution map.
- Create table spaces in the database partition group.

You cannot do the following operations while the utility is running:

- Start another redistribution operation on the database partition group
- Execute an ALTER TABLE statement on any table in the database partition group
- Drop the database partition group
- Alter the database partition group.

You cannot use this procedure to redistribute data after adding a database partition to a single-partition system unless all affected tables have a distribution key. The REDISTRIBUTE DATABASE PARTITION GROUP command relies on distribution keys to redistribute data. The distibution key is generated automatically when a table is created in a multi-partition database partition group, or can be explicitly defined using the CREATE TABLE or ALTER TABLE SQL statements. If your tables were created in a single-partition partition group, and you did not define the distribution key in the CREATE TABLE SQL statement, there will be no distribution keys defined. You must use the ALTER TABLE SQL statement to create a distribution key for each affected table before redistributing the data.

Procedure:

To redistribute data across database partitions in a database partition group:

1. Connect to the database partition that contains the system catalog tables.
2. Perform prerequisite tasks, if necessary.
3. Issue the REDISTRIBUTE DATABASE PARTITION GROUP command.

   Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

   Specify the following arguments:

   database partition group name
   You must specify the database partition group within which data is to be redistributed.

   UNIFORM
   If data is evenly distributed and is to remain evenly distributed, either specify UNIFORM or omit any distribution-type argument. UNIFORM is the default.

   USING DISTFILE distfile-name
   To specify a custom distribution that corrects or creates data skew, include the distribution file name. The Redistribute Data utility uses this file to construct a target distribution map.

   USING TARGETMAP targetmap-name
   The Redistribute Data utility uses the specified target map directly.

   For details, refer to the REDISTRIBUTE DATABASE PARTITION GROUP command-line utility information.
4. After redistribution is complete:
   - Recreate any replicated materialized query tables dropped before redistribution.

- Execute the RUNSTATS command to collect data distribution statistics for the SQL compiler and optimizer to use when it chooses data access plans for queries.

Note: The Explain tables contain information about the distribution map used to redistribute data.

You can also call the sqludrdt API from a client application to redistribute data across a database partition group.

Related concepts:

- "Data redistribution" on page 343
- "Log space requirements for data redistribution" on page 347
- "Redistribution error recovery" on page 350

Related tasks:

- "Determining whether to redistribute data" on page 344

Related reference:

- "Restrictions on native XML data store" in *XML Guide*
- "sqludrdt API - Redistribute data across a database partition group" in *Administrative API Reference*

Log space requirements for data redistribution

Before you redistribute data across database partitions, consider the log-space requirements.

The log must be large enough to accommodate the INSERT and DELETE operations at each database partition where data is being redistributed. The heaviest logging requirements will be either on the database partition that will lose the most data, or on the database partition that will gain the most data.

If you are moving to a larger number of database partitions, use the ratio of current database partitions to the new number of database partitions to estimate the number of INSERT and DELETE operations. For example, consider redistributing data that is uniformly distributed before redistribution. If you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will move to the new database partition. This means that twenty percent of the DELETE operations will occur on each of the four original database partitions, and all of the INSERT operations will occur on the new database partition.

Consider a non-uniform distribution of the data, such as the case in which the distribution key contains many NULL values. In this case, all rows that contain a NULL value in the distribution key move from one database partition under the old distribution scheme and to a different database partition under the new distribution scheme. As a result, the amount of log space required on those two database partitions increases, perhaps well beyond the amount calculated by assuming uniform distribution.

The redistribution of each table is a single transaction. For this reason, when you estimate log space, you multiply the percentage of change, such as twenty percent, by the size of the largest table. Consider, however, that the largest table might be

uniformly distributed but the second largest table, for example, might have one or more inflated database partitions. In such a case, consider using the non-uniformly distributed table instead of the largest one.

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 256 GB, then the data redistribution must be done in steps. Use the "makepmap" utility to generate a series of target distribution maps, one for each step. You might also set the *logsecond* database configuration parameter to -1 to avoid most log space problems.

Related concepts:

- "Data redistribution" on page 343

Redistributing data using step-wise redistribute procedures

The stepwise redistribute stored procedures can be used to safely redistribute a database partition group in a number of steps:

1. Analyze the database partition group regarding log space availability and data skew using ANALYZE_LOG_SPACE procedure – Retrieve log space analysis information.

   The analyze_log_space function returns a result set (an open cursor) of the log space analysis results, containing fields for each of the database partitions of the given database partition group.
2. Create data distribution file for a given table using GENERATE_DISTFILE procedure – Generate a data distribution file

   The generate_Distfile function generates a data distribution file for the given table and saves it under the given fileName.
3. Create and report the content of a stepwise redistribution plan for the database partition group using STEPWISE_REDISTRIBUTE_DBPG procedure – Redistribute part of database partition group
4. Create data distribution file for a given table using GET_SWRD_SETTINGS procedure – Retrieve redistribute information and SET_SWRD_SETTINGS procedure – Create or change redistribute registry.

   The get_swrd_settings function reads the existing redistribute registry records for the given database partition group.

   The set_swrd_settings function creates or make changes to the redistribute registry. If the registry does not exist, it creates it and add records into it. If the registry already exists, it uses overwriteSpec to identify which of the field values need to be overwritten. The overwriteSpec field enables this function to take NULL inputs for the fields that do not need to be updated.
5. Redistribute the database partition group according to the plan using STEPWISE_REDISTRIBUTE_DBPG procedure – Redistribute part of database partition group.

   The stepwise_redistribute_dbpg function redistributes part of the database partition group according to the input and the setting file.

Usage example

The following is an example of a CLP script on AIX:

```
# ------------------------------------------------------------------------------
# Set the database you wish to connect to
# ------------------------------------------------------------------------------
dbName="SAMPLE"

# ------------------------------------------------------------------------------
# Set the target database partition group name
# ------------------------------------------------------------------------------
dbpgName="IBMDEFAULTGROUP"

# ------------------------------------------------------------------------------
# Specify the table name and schema
# ------------------------------------------------------------------------------
tbSchema="$USER"
tbName="STAFF"

# ------------------------------------------------------------------------------
# Specify the name of the data distribution file
# ------------------------------------------------------------------------------
distFile="$HOME/sqllib/function/$dbName.IBMDEFAULTGROUP_swrdData.dst"

export DB2INSTANCE=$USER
export DB2COMM=TCPIP

# ------------------------------------------------------------------------------
# Invoke call statements in clp
# ------------------------------------------------------------------------------
db2start
db2 -v "connect to $dbName"

# ------------------------------------------------------------------------------
# Analysing the effect of adding a database partition without applying the changes - a 'what if'
# hypothetical analysis
#
# - In the following case, the hypothesis is adding database partition 40, 50 and 60 to the
#   database partition group, and for database partitions 10,20,30,40,50,60, using a respective
#   target ratio of 1:2:1:2:1:2.
#
# NOTE: in this example only partitions 10, 20 and 30 actually exist in the database
#       partition group
# ------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'A', '40,50,60', '10,20,30,40,50,60', '1,2,1,2,1,2')"

# ------------------------------------------------------------------------------
# Analysing the effect of dropping a database partition without applying the changes
#
# - In the following case, the hypothesis is dropping database partition 30 from the database
#   partition group, and redistributing the data in database partitions 10 and 20 using a
#   respective target ratio of 1 : 1
#
# NOTE: In this example all database partitions 10, 20 and 30 should exist in the database
#       partition group
# ------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'D', '30', '10,20','1,1')"

# ------------------------------------------------------------------------------
# Generate a data distribution file to be used by the redistribute process
# ------------------------------------------------------------------------------
db2 -v "call sysproc.generate_distfile('$tbSchema', '$tbName', '$distFile')"

# ------------------------------------------------------------------------------
# Write a step wise redistribution plan into a registry
#
# Setting the 10th parameter to 1, may cause a currently running step wise redistribute
# stored procedure to complete the current step and stop, until this parameter is reset
```

```
# to 0, and the redistribute stored procedure is called again.
# --------------------------------------------------------------------------------
db2 -v "call sysproc.set_swrd_settings('$dbpgName', 255, 0, ' ', '$distFile', 1000,
12, 2, 1, 0, '10,20,30', '50,50,50')"

# --------------------------------------------------------------------------------
# Report the content of the step wise redistribution plan for the given database
# partition group.
# --------------------------------------------------------------------------------
db2 -v "call sysproc.get_swrd_settings('$dbpgName', 255, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"

# --------------------------------------------------------------------------------
# Redistribute the database partition group "dbpgName" according to the redistribution
# plan stored in the registry by set_swrd_settings. It starting with step 3 and
# redistributes the data until 2 steps in the redistribution plan are completed.
# --------------------------------------------------------------------------------
db2 -v "call sysproc.stepwise_redistribute_dbpg('$dbpgName', 3, 2)"
```

Related concepts:

- "Data redistribution" on page 343

Redistribution error recovery

After the redistribution operation begins to execute, a file is written to the `redist` subdirectory of the `sqllib` directory. This status file lists any operations that are done on database partitions, the names of the tables that were redistributed, and the completion status of the operation. If a table cannot be redistributed, its name and the applicable SQLCODE is listed in the file. If the redistribution operation cannot begin because of an incorrect input parameter, the file is not written and an SQLCODE is returned.

The file has the following naming convention:

```
For UNIX platforms:
  databasename.database partition groupname.timestamp
For non-UNIX platforms:
  databasename\database partition groupname\date\time
```

Note: On non-UNIX platforms, only the first eight (8) bytes of the database partition group name are used.

If the data redistribution operation fails, some tables may be redistributed, while others are not. This occurs because data redistribution is performed a table at a time. You have two options for recovery:

- Use the CONTINUE option to continue the operation to redistribute the remaining tables.
- Use the ROLLBACK option to undo the redistribution and set the redistributed tables back to their original state. The rollback operation can take about the same amount of time as the original redistribution operation.

Before you can use either option, a previous data redistribution operation must have failed such that the REBALANCE_PMID column in the SYSCAT.DBPARTITIONGROUPS table is set to a non-NULL value.

If you happen to delete the status file by mistake, you can still attempt a CONTINUE operation.

Related concepts:

- "Data redistribution" on page 343

Related tasks:

- "Redistributing data across database partitions" on page 345

Chapter 15. The database system monitor information

The DB2 database manager maintains data about its operation, its performance, and the applications using it. This data is maintained as the database manager runs, and can provide important performance and troubleshooting information. For example, you can find out:

- The number of applications connected to a database, their status, and which SQL and XQuery statements each application is executing, if any.
- Information that shows how well the database manager and database are configured, and helps you to tune them.
- When deadlocks occurred for a specified database, which applications were involved, and which locks were in contention.
- The list of locks held by an application or a database. If the application cannot proceed because it is waiting for a lock, there is additional information on the lock, including which application is holding it.

Because collecting some of this data introduces overhead on the operation of DB2, **monitor switches** are available to control which information is collected. To set monitor switches explicitly, use the UPDATE MONITOR SWITCHES command or the sqlmon() API. (You must have SYSADM, SYSCTRL, SYSMAINT, or SYSMON authority.)

You can access the data that the database manager maintains either by taking a snapshot or by using an event monitor.

Taking a snapshot

You can take a snapshot in one of the following three ways:

- Use the GET SNAPSHOT command from the command line.
- Write your own application, using the db2GetSnapshot() API call.
- Use snapshot table functions to return monitor data about a specific area of the database system.

Using an event monitor

An event monitor captures system monitor information after particular events have occurred, such as the end of a transaction, the end of a statement, or the detection of a deadlock. This information can be written to files or to a named pipe.

To use an event monitor:

1. Create its definition with the Control Center or the SQL statement CREATE EVENT MONITOR. This statement stores the definition in database system catalogs.
2. Activate the event monitor through the Control Center, or with the SQL statement:

   ```
   SET EVENT MONITOR evname STATE 1
   ```

   If writing to a named pipe, start the application reading from the named pipe before activating the event monitor. You can either write your own application to do this, or use **db2evmon**. Once the event monitor is active and starts

writing events to the pipe, **db2evmon** will read them as they are being generated and write them to standard output.

3. Read the trace. If using a file event monitor, you can view the binary trace that it creates in either of the following ways:
   - Use the **db2evmon** tool to format the trace to standard output.
   - Click on the **Event Analyzer** icon in the Control Center on a Windows-based operating system to use a graphical interface to view the trace, search for keywords, and filter out unwanted data.

     Note: If the database system that you are monitoring is not running on the same machine as the Control Center, you must copy the event monitor file to the same machine as the Control Center before you can view the trace. An alternative method is to place the file in a shared file system accessible to both machines.

Performance impact when using a deadlock event monitor

When a deadlock event monitor is active with the HISTORY option enabled, the general performance of a DB2 database system is impacted in the following ways:

- Memory used in the package cache for cached dynamic SQL and XQuery statements that are listed in the the statement history is not released until that particular statement history is no longer needed (that is, the current unit of work ends). This can cause the package cache size to grow due to the increased use of space in the cache that cannot be freed.
- There is a minor impact in system performance, resulting from the copying of statement information to the statement history list.
- There is an increased use of the DB2 system monitor heap, at each database partition, in order to keep a statement history list for each active application at that database partition. The amount of the increase depends on the number of statements executed in each unit of work by each application. A suggested computation for monitor heap follows:

```
If an event monitor is of type DEADLOCK
   and the WITH DETAILS HISTORY option is running,
add X*100 bytes times the maximum number of concurrent applications
   you expect to be running,
where X is the expected maximum number of statements in your
   application's unit of work.
If the event monitor is of type DEADLOCK
   and the WITH DETAILS HISTORY VALUES option is running,
also add X*Y bytes times the maximum number of concurrent applications
   you expect to be running,
where Y is the sum of the expected maximum size of parameter values being
   bound into your SQL and XQuery statements.
```

When a deadlock event monitor is active with the VALUES option enabled, the general performance of a DB2 database system is impacted in the following ways (in addition to the ones listed previously for the HISTORY option):

- There is a very minor impact in system performance resulting from the copying of statement information to the statement history list.
- There is an increased use of the DB2 system monitor heap at each database partition in order to keep a statement history list for each active application at that database partition. The increase depends on the number of data values used per statement as well as the number of statements executed in each unit of work by each application.

- The database manager maintains an extra copy of the data values which, depending on the size and number of the variables, may have an impact on performance.

The memory impact on the DB2 system monitor heap can become substantial when both the HISTORY and the VALUES options are specified for the deadlock event monitor. To decrease the impact, use these options only when they are needed. Another way to decrease the impact is to increase the configured size of the DB2 System Monitor heap at all database partitions prior to enabling the event monitor.

When an actual deadlock occurs and there is a deadlock event monitor active, system performance is impacted by the generation of the event monitor records. The degree of impact and its duration depends on the number of applications and database partitions involved in the deadlock as well as the number of statements and data values in the relevant statement history lists.

Related concepts:

- "Quick-start tips for performance tuning" on page 6

Chapter 16. Measuring performance through benchmarks

Benchmark testing

Benchmark testing is a normal part of the application development life cycle. It is a team effort that involves both application developers and database administrators (DBAs), and should be performed against your application in order to determine current performance and improve it. If the application code has been written as efficiently as possible, additional performance gains might be realized from tuning the database and database manager configuration parameters. You can even tune application parameters to meet the requirements of the application better.

You run different types of benchmark tests to discover specific kinds of information:

- A *transaction per second* benchmark determines the throughput capabilities of the database manager under certain limited laboratory conditions.
- An *application* benchmark tests the same throughput capabilities under conditions that are closer production conditions.

Benchmarking tuning configuration parameters is based upon these "real-world" conditions, and requires repeatedly running SQL taken from your application with varying parameter values until the application runs as efficiently as possible.

The benchmarking methods described here are oriented toward tuning configuration parameters. However, the same basic technique can be used for tuning other factors that affect performance, such as:

- SQL statements
- Indexes
- Table space configuration
- Application code
- Hardware configuration.

Benchmarking is helpful in understanding how the database manager responds under varying conditions. You can create scenarios that test deadlock handling, utility performance, different methods of loading data, transaction rate characteristics as more users are added, and even the effect on the application of using a new release of the product.

Benchmark testing methods

Benchmark tests are based on a repeatable environment so that the same test run under the same conditions will yield results that you can legitimately compare.

You might begin benchmarking by running the test application in a normal environment. As you narrow down a performance problem, you can develop specialized test cases that limit the scope of the function that you are testing. The specialized test cases need not emulate an entire application to obtain valuable information. Start with simple measurements, and increase the complexity only when necessary.

Characteristics of good benchmarks or measurements include:

- Tests are repeatable.

- Each iteration of a test starts in the same system state.
- No other functions or applications are active in the system unless the scenario includes some amount of other activity going on in the system.

  Note: Started applications use memory even when they are minimized or idle. This increases the probability that paging will skew the results of the benchmark and violates the repeatability rule.
- The hardware and software used for benchmarking match your production environment.

For benchmarking, you create a scenario and then applications in this scenario several times, capturing key information during each run. Capturing key information after each run is of primary importance in determining the changes that might improve performance of both the application and the database.

Related concepts:

- "Benchmark preparation" on page 358
- "Benchmark test analysis example" on page 373
- "Benchmark test creation" on page 359
- "Benchmark test execution" on page 371

Benchmark preparation

Complete the logical design of the database against which the application runs before you start performance benchmarking. Set up and populate tables, views, and indexes. Normalize tables, bind application packages, and populate tables with realistic data.

You should also have determined the final physical design of the database. Place database manager objects in their final disk locations, size log files, determining the location of work files and backup, and test backup procedures. In addition, check packages to make sure that performance options such as row blocking are enabled when possible.

You should have reached a point in application programming and testing phases that will enable you to create your benchmark programs. Although the practical limits of an application might be revealed during the benchmark testing, the purpose of the benchmark described here is to measure performance, not to detect defects or abends.

Your benchmarking test program will need to run in as accurate a representation of the final production environment as possible. Ideally, it should run on the same model of server with the same memory and disk configurations. This is especially important when the application will ultimately involve large numbers of users and large amounts of data. The operating system itself and any communications or file-serving facilities used directly by the benchmark should also have been tuned.

Make sure that you run benchmark tests with a production-size database. An individual SQL statement should return as much data and require as much sorting as in production. This rule ensures that the application will test representative memory requirements.

SQL statements to be benchmarked should be either *representative* or *worst-case*, as described below:

Representative SQL

Representative SQL includes those statements that are executed during typical operations of the application being benchmarked. The statements that are selected will depend on the nature of the application. For example, a data-entry application might test an INSERT statement, while a banking transaction might test a FETCH, an UPDATE, and several INSERTs. Consider the frequency of execution and volume of data processed by the statements chosen average. If the volumes are excessive, consider the statements under the *worst-case* category, even if they are typical SQL statements.

Worst-case SQL

Statements falling in this category include:

- Statements that are executed frequently.
- Statements that have high volumes of data being processed.
- Statements that are time-critical.

  For example, an application that is run when a telephone call is received from a customer and the statements must be run to retrieve and update the customer's information while the customer is waiting.
- Statements with the largest number of tables being joined or with the most complex SQL in the application.

  For example, a banking application that produces combined customer statements of monthly activity for all their different types of accounts. A common table may list customer address and account numbers; however, several other tables must be joined to process and integrate all of the necessary account transaction information. Multiply the work necessary for one account by the several thousand accounts that must be processed during the same period, and the potential time savings drives the performance requirements.
- Statements that have a poor access path, such as one that is not executed very often and is not supported by the indexes that have been created for the tables involved.
- Statements that have a long elapsed time.
- A statement that is only executed at application initialization but has disproportionate resource requirements.

  For example, an application that generates a list of account work that must be processed during the day. When the application is started, the first major SQL statement causes a 7-way join, which creates a very large list of all the accounts for which this application user is responsible. The statement might only be run a few times per day, but takes several minutes to run when it has not been tuned properly.

Related concepts:

- "Benchmark test creation" on page 359
- "Benchmark testing" on page 357

Benchmark test creation

Consider a variety of factors when you design and implement a benchmark program. Because the main purpose of the program is to simulate a user application, the overall structure of the program varies. You might use the entire application as the benchmark and simply introduce a means for timing the SQL statements to be analyzed. For large or complex applications, it might be more practical to include only blocks that contain the important statements.

To test the performance of specific SQL statements, you might include these statements alone in the benchmark program along with the necessary CONNECT, PREPARE, OPEN, and other statements and a timing mechanism.

Another factor to consider is the type of benchmark to use. One option is to run a set of SQL statements repeatedly over a time interval. The ratio of the number of statements executed and this time interval would give the throughput for the application. Another option is simply to determine the time required to execute individual SQL statements.

For all benchmark testing, you need an efficient timing system to calculate the elapsed time, whether for individual SQL statements or the application as a whole. To simulate applications in which individual SQL statements are executed in isolation, it might be important to track times for CONNECT, PREPARE, and COMMIT statements. However, for programs that process many different statements, perhaps only a single CONNECT or COMMIT is necessary, and focusing on just the execution time for an individual statement might be the priority.

Although the elapsed time for each query is an important factor in performance analysis, it might not necessarily reveal bottlenecks. For example, information on CPU usage, locking, and buffer pool I/O might show that the application is I/O bound and is not using the CPU to its full capacity. A benchmark program should allow you to obtain this kind of data for a more detailed analysis if needed.

Not all applications send the entire set of rows retrieved from a query to some output device. For example, the whole answer set might be input for another program, so that none of the rows from the first application are sent as output. Formatting data for screen output usually has high CPU cost and might not reflect user need. To provide an accurate simulation, a benchmark program should reflect the row handling of the specific application. If rows are sent to an output device, inefficient formatting could consume the majority of CPU processing time and misrepresent the actual performance of the SQL statement itself.

The db2batch Benchmark Tool: A benchmark tool (`db2batch`) is provided in the `bin` subdirectory of your instance `sqllib` directory. This tool uses many of guidelines for creating a benchmark program. This tool can read SQL statements from either a flat file or standard input, dynamically describe and prepare the statements, and return an answer set. It also allows you to control the size of the answer set, as well as the number of rows that are sent from this answer set to an output device.

You can specify the level of performance-related information supplied, including the elapsed time, CPU and buffer pool usage, locking, and other statistics collected from the database monitor. If you are timing a set of SQL statements, `db2batch` also summarizes the performance results and provides both arithmetic and geometric means. For syntax and options, type `db2batch -h` on a command line.

Examples of db2batch tests

The following example shows how `db2batch` could be used with an input file `db2batch.sql`:

```
-- db2batch.sql
-- ------------
--#SET PERF_DETAIL 3
--#SET ROWS_OUT 5


-- This query lists employees, the name of their department
-- and the number of activities to which they are assigned for
-- employees who are assigned to more than one activity less than
-- full-time.
--#COMMENT Query 1
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2;
--#SET PERF_DETAIL 1
--#SET ROWS_OUT 5

--#COMMENT Query 2
select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2;
```

Figure 33. Sample Benchmark Input File: db2batch.sql

Using the following invocation of the benchmark tool:

```
db2batch -d sample -f db2batch.sql
```

Produces the following output:

```
* Timestamp: Thu Feb 02 2006 16:03:17 EST
* Comment: " Query 1"
-------------------------------------------

* SQL Statement Number 1:

select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) > 2;
```

Figure 34. Sample Output From db2batch (Part 1)

```
LASTNAME        FIRSTNME     DEPTNAME                       NUM_ACT
--------------- ------------ ------------------------------ -----------
JEFFERSON       JAMES        ADMINISTRATION SYSTEMS                   3
JOHNSON         SYBIL        ADMINISTRATION SYSTEMS                   4
NICHOLLS        HEATHER      INFORMATION CENTER                       4
PEREZ           MARIA        ADMINISTRATION SYSTEMS                   4
SMITH           DANIEL       ADMINISTRATION SYSTEMS                   7

* 5 row(s) fetched, 5 row(s) output.

* Elapsed Time is:       0.046319 seconds

            Monitoring Information

Instance name                                 = dyuping
Node name                                     =
Node type                                     = Database Server with local clients
Snapshot timestamp                            = 02/02/2006 16:03:18.188050
```

Figure 35. Sample Output From db2batch (Part 1)

```
            Database Manager Snapshot

Number of database partitions in DB2 instance  = 1
Database manager status                        = Active
Product name                                   = DB2 v9.1.0.0
Service level                                  = n060128

Private Sort heap allocated                    = 0
Private Sort heap high water mark              = 0
Post threshold sorts                           = 0
Piped sorts requested                          = 1
Piped sorts accepted                           = 1

Start Database Manager timestamp               = 02/01/2006 19:07:35.301696
Last reset timestamp                           = 02/02/2006 16:03:18.138248
Remote connections to db manager               = 4
Remote connections executing in db manager     = 0
Local connections                              = 2
Local connections executing in db manager      = 0
Active local databases                         = 2

High water mark for agents registered          = 9
High water mark for agents waiting for a token = 0
Agents registered                              = 9
Agents waiting for a token                     = 0
Idle agents                                    = 0

Committed private Memory (Bytes)               = 1572864

Switch list for db partition number 0
Buffer Pool Activity Information  (BUFFERPOOL) = ON  02/02/2006 16:03:18.071139
Lock Information                       (LOCK) = ON  02/02/2006 13:45:50.679836
Sorting Information                    (SORT) = ON  02/02/2006 16:03:18.071143
SQL Statement Information          (STATEMENT) = ON  02/02/2006 16:03:18.071136
Table Activity Information             (TABLE) = ON  02/02/2006 16:03:18.071138
Take Timestamp Information         (TIMESTAMP) = ON  02/01/2006 19:07:35.301696
Unit of Work Information                 (UOW) = ON  02/02/2006 16:03:18.071133


Agents assigned from pool                      = 1578
Agents created from empty pool                 = 10
Agents stolen from another application         = 0
High water mark for coordinating agents        = 9
Max agents overflow                            = 0
Hash joins after heap threshold exceeded       = 0

Total number of gateway connections            = 0
Current number of gateway connections          = 0
Gateway connections waiting for host reply     = 0
Gateway connections waiting for client request = 0
Gateway connection pool agents stolen          = 0
```

```
Memory usage for database manager:

    Memory Pool Type                          = Database Monitor Heap
       Current size (bytes)                   = 655360
       High water mark (bytes)                = 655360
       Configured size (bytes)                = 393216


    Memory Pool Type                          = Other Memory
       Current size (bytes)                   = 8323072
       High water mark (bytes)                = 8323072
       Configured size (bytes)                = 45088768
            Application Snapshot

Application handle                         = 339
Application status                         = UOW Waiting
Status change time                         = 02/02/2006 16:03:18.185731
Application code page                      = 819
Application country/region code            = 1
DUOW correlation token                     = *LOCAL.dyuping.060202210317
Application name                           = db2batch
Application ID                             = *LOCAL.dyuping.060202210317
Sequence number                            = 00001
TP Monitor client user ID                  =
TP Monitor client workstation name         =
TP Monitor client application name         = DB2BATCH
TP Monitor client accounting string        =

CONNECT Authorization ID                   = DYUPING
Client login ID                            = dyuping
Configuration NNAME of client              = honcho
Client database manager product ID         = SQL09010
Process ID of client application           = 2543638
Platform of client application             = AIX 64BIT
Communication protocol of client           = Local Client

Database name                              = SAMPLE
Database path                              = /home/dyuping/dyuping/NODE0000/SQL00003/
Client database alias                      = SAMPLE
Input database alias                       = SAMPLE
The highest authority level granted        =
        Direct DBADM authority
        Direct CREATETAB authority
        Direct BINDADD authority
        Direct CONNECT authority
        Direct CREATE_NOT_FENC authority
        Direct LOAD authority
        Direct IMPLICIT_SCHEMA authority
        Direct CREATE_EXT_RT authority
        Direct QUIESCE_CONN authority
        Indirect SYSADM authority
        Indirect CREATETAB authority
        Indirect BINDADD authority
        Indirect CONNECT authority
        Indirect IMPLICIT_SCHEMA authority


Coordinating database partition number     = 0
Coordinator agent process or thread ID     = 1626226

Agents associated with the application     = 1


Connection request start timestamp         = 02/02/2006 16:03:17.581904
Connect request completion timestamp       = 02/02/2006 16:03:18.045845
Application idle time                      = 0
Inbound communication address              = *LOCAL.dyuping

Last reset timestamp                       = 02/02/2006 16:03:18.138248
Agents stolen                              = 0
Agents waiting on locks                    = 0
Maximum associated agents                  = 1
Priority at which application agents work  = 0
Priority type                              = Dynamic
Locks held by application                  = 3
Lock waits since connect                   = 0
```

```
Time application waited on locks (ms)      = 0
Deadlocks detected                         = 0
Lock escalations                           = 0
Exclusive lock escalations                 = 0
Number of Lock Timeouts since connected    = 0
Total time UOW waited on locks (ms)        = 0

Total sorts                                = 1
Total sort time (ms)                       = 0
Total sort overflows                       = 0

Data pages copied to extended storage      = 0
Index pages copied to extended storage     = 0
Data pages copied from extended storage    = 0
Index pages copied from extended storage   = 0
Buffer pool data logical reads             = 43
Buffer pool data physical reads            = 10
Buffer pool temporary data logical reads   = 0
Buffer pool temporary data physical reads  = 0
Buffer pool data writes                    = 0
Buffer pool index logical reads            = 7
Buffer pool index physical reads           = 1
Buffer pool temporary index logical reads  = 0
Buffer pool temporary index physical reads = 0
Buffer pool index writes                   = 0
Buffer pool xda logical reads              = 0
Buffer pool xda physical reads             = 0
Buffer pool temporary xda logical reads    = 0
Buffer pool temporary xda physical reads   = 0
Buffer pool xda writes                     = 0
Total buffer pool read time (milliseconds) = 9
Total buffer pool write time (milliseconds)= 0
Time waited for prefetch (ms)              = 0
Unread prefetch pages                      = 0
Direct reads                               = 22
Direct writes                              = 0
Direct read requests                       = 5
Direct write requests                      = 0
Direct reads elapsed time (ms)             = 7
Direct write elapsed time (ms)             = 0

Number of SQL requests since last commit   = 7
Commit statements                          = 0
Rollback statements                        = 0
Dynamic SQL statements attempted           = 2
Static SQL statements attempted            = 0
Failed statement operations                = 0
Select SQL statements executed             = 1
Xquery statements executed                 = 0
Update/Insert/Delete statements executed   = 0
DDL statements executed                    = 0
Internal automatic rebinds                 = 0
Internal rows deleted                      = 0
Internal rows inserted                     = 0
Internal rows updated                      = 0
Internal commits                           = 0
Internal rollbacks                         = 0
Internal rollbacks due to deadlock         = 0
Binds/precompiles attempted                = 0
Rows deleted                               = 0
Rows inserted                              = 0
Rows updated                               = 0
Rows selected                              = 5
Rows read                                  = 129
Rows written                               = 0

UOW log space used (Bytes)                 = 0
Previous UOW completion timestamp          = 02/02/2006 16:03:18.045845
Elapsed time of last completed uow (sec.ms)= 0.000000
UOW start timestamp                        = 02/02/2006 16:03:18.072281
UOW stop timestamp                         = 12/31/1969 19:00:00.000000
UOW completion status                      = 0

Open remote cursors                        = 0
Open remote cursors with blocking          = 0
Rejected Block Remote Cursor requests      = 0
```

```
Accepted Block Remote Cursor requests      = 2
Open local cursors                         = 0
Open local cursors with blocking           = 0
Total User CPU Time used by agent (s)      = 0.020000
Total System CPU Time used by agent (s)    = 0.010000
Host execution elapsed time                = 0.045308

Package cache lookups                      = 1
Package cache inserts                      = 1
Application section lookups                = 2
Application section inserts                = 1
Catalog cache lookups                      = 6
Catalog cache inserts                      = 3
Catalog cache overflows                    = 0
Catalog cache heap full                    = 0
Catalog cache high water mark              = 0


Workspace Information
 Shared high water mark                    = 0
 Total shared overflows                    = 0
 Total shared section inserts              = 0
 Total shared section lookups              = 0
 Private high water mark                   = 25848
 Total private overflows                   = 0
 Total private section inserts             = 1
 Total private section lookups             = 1

Number of hash joins                       = 2
Number of hash loops                       = 0
Number of hash join overflows              = 0
Number of small hash join overflows        = 0


Statement type                             = Dynamic SQL Statement
Statement                                  = Close
Section number                             = 4
Application creator                        = NULLID
Package name                               = SYSSN200
Consistency Token                          = SYSLVL01
Package Version ID                         =
Cursor name                                = SQL_CURSN200C4
Statement database partition number        = 0
Statement start timestamp                  = 02/02/2006 16:03:18.178967
Statement stop timestamp                   = 02/02/2006 16:03:18.185705
Elapsed time of last completed stmt(sec.ms)= 0.000124
Total Statement user CPU time              = 0.010000
Total Statement system CPU time            = 0.000000
SQL compiler cost estimate in timerons     = 87
SQL compiler cardinality estimate          = 13
Degree of parallelism requested            = 1
Number of agents working on statement      = 0
Number of subagents created for statement  = 1
Statement sorts                            = 1
Total sort time                            = 0
Sort overflows                             = 0
Rows read                                  = 116
Rows written                               = 0
Rows deleted                               = 0
Rows updated                               = 0
Rows inserted                              = 0
Rows fetched                               = 5
Buffer pool data logical reads             = 4
Buffer pool data physical reads            = 1
Buffer pool temporary data logical reads   = 0
Buffer pool temporary data physical reads  = 0
Buffer pool index logical reads            = 0
Buffer pool index physical reads           = 0
Buffer pool temporary index physical reads = 0
Buffer pool xda logical reads              = 0
Buffer pool xda physical reads             = 0
Buffer pool temporary xda logical reads    = 0
Buffer pool temporary xda physical reads   = 0
Blocking cursor                            = YES
Dynamic SQL statement text:select lastname, firstnme,
deptname, count(*) as num_act from employee, department, emp_act
where employee.workdept = department.deptno and
```

```
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname having count(*) > 2

Agent process/thread ID                     = 1626226

  Memory usage for agent:

    Memory Pool Type                          = Other Memory
       Current size (bytes)                   = 65536
       High water mark (bytes)                = 65536
       Configured size (bytes)                = 4294967296


    Memory Pool Type                          = Application Heap
       Current size (bytes)                   = 262144
       High water mark (bytes)                = 327680
       Configured size (bytes)                = 1245184

    Memory Pool Type                          = Application Control Heap
       Current size (bytes)                   = 65536
       High water mark (bytes)                = 65536
       Configured size (bytes)                = 655360


             Database Snapshot

Database name                               = SAMPLE
Database path                               = /home/dyuping/dyuping/NODE0000/SQL00003/
Input database alias                        = SAMPLE
Database status                             = Active
Catalog database partition number           = 0
Catalog network node name                   =
Operating system running at database server= AIX 64BIT
Location of the database                    = Local
First database connect timestamp            = 02/02/2006 16:03:17.581904
Last reset timestamp                        = 02/02/2006 16:03:18.138248
Last backup timestamp                       = 12/31/1969 19:00:00.000000
High water mark for connections             = 4
Application connects                        = 4
Secondary connects total                    = 3
Applications connected currently            = 4
Appls. executing in db manager currently    = 0
Agents associated with applications         = 3
Maximum agents associated with applications= 4
Maximum coordinating agents                 = 4

Locks held currently                        = 3
Lock waits                                  = 0
Time database waited on locks (ms)          = 0
Lock list memory in use (Bytes)             = 5472
Deadlocks detected                          = 0
Lock escalations                            = 0
Exclusive lock escalations                  = 0
Agents currently waiting on locks           = 0
Lock Timeouts                               = 0
Number of indoubt transactions              = 0

Total Private Sort heap allocated           = 0
Total Shared Sort heap allocated            = 0
Shared Sort heap high water mark            = 59
Total sorts                                 = 1
Total sort time (ms)                        = 0
Sort overflows                              = 0
Active sorts                                = 0

Buffer pool data logical reads              = 43
Buffer pool data physical reads             = 10
Buffer pool temporary data logical reads    = 0
Buffer pool temporary data physical reads   = 0
Asynchronous pool data page reads           = 0
Buffer pool data writes                     = 0
Asynchronous pool data page writes          = 0
Buffer pool index logical reads             = 7
Buffer pool index physical reads            = 1
Buffer pool temporary index logical reads   = 0
```

```
Buffer pool temporary index physical reads = 0
Asynchronous pool index page reads         = 0
Buffer pool index writes                   = 0
Asynchronous pool index page writes        = 0
Buffer pool xda logical reads              = 0
Buffer pool xda physical reads             = 0
Buffer pool temporary xda logical reads    = 0
Buffer pool temporary xda physical reads   = 0
Buffer pool xda writes                     = 0
Asynchronous pool xda page reads           = 0
Asynchronous pool xda page writes          = 0
Total buffer pool read time (milliseconds) = 9
Total buffer pool write time (milliseconds)= 0
Total elapsed asynchronous read time       = 0
Total elapsed asynchronous write time      = 0
Asynchronous data read requests            = 0
Asynchronous index read requests           = 0
Asynchronous xda read requests             = 0
LSN Gap cleaner triggers                   = 0
Dirty page steal cleaner triggers          = 0
Dirty page threshold cleaner triggers      = 0
Time waited for prefetch (ms)              = 0
Unread prefetch pages                      = 0
Direct reads                               = 22
Direct writes                              = 0
Direct read requests                       = 5
Direct write requests                      = 0
Direct reads elapsed time (ms)             = 7
Direct write elapsed time (ms)             = 0
Database files closed                      = 0
Data pages copied to extended storage      = 0
Index pages copied to extended storage     = 0
Data pages copied from extended storage    = 0
Index pages copied from extended storage   = 0
Vectored IOs                               = Not Collected
Pages from vectored IOs                    = Not Collected
Block IOs                                  = Not Collected
Pages from block IOs                       = Not Collected
Physical page maps                         = Not Collected


Host execution elapsed time                = 0.045308

Commit statements attempted                = 0
Rollback statements attempted              = 0
Dynamic statements attempted               = 2
Static statements attempted                = 0
Failed statement operations                = 0
Select SQL statements executed             = 1
Xquery statements executed                 = 0
Update/Insert/Delete statements executed   = 0
DDL statements executed                    = 0

Internal automatic rebinds                 = 0
Internal rows deleted                      = 0
Internal rows inserted                     = 0
Internal rows updated                      = 0
Internal commits                           = 0
Internal rollbacks                         = 0
Internal rollbacks due to deadlock         = 0

Rows deleted                               = 0
Rows inserted                              = 0
Rows updated                               = 0
Rows selected                              = 5
Rows read                                  = 129
Binds/precompiles attempted                = 0

Log space available to the database (Bytes)= 20400000
Log space used by the database (Bytes)     = 0
Maximum secondary log space used (Bytes)   = 0
Maximum total log space used (Bytes)       = 0
Secondary logs allocated currently         = 0
Log pages read                             = 0
Log pages written                          = 0
Appl id holding the oldest transaction     = 339
```

```
Package cache lookups                   = 1
Package cache inserts                   = 1
Package cache overflows                 = 0
Package cache high water mark (Bytes)   = 720896
Application section lookups             = 2
Application section inserts             = 1

Catalog cache lookups                   = 6
Catalog cache inserts                   = 3
Catalog cache overflows                 = 0
Catalog cache heap full                 = 0
Catalog cache high water mark           = 65536

Workspace Information
 Shared high water mark                 = 0
 Corresponding shared overflows         = 0
 Total shared section inserts           = 0
 Total shared section lookups           = 0
 Private high water mark                = 25848
 Corresponding private overflows        = 0
 Total private section inserts          = 1
 Total private section lookups          = 1

Number of hash joins                    = 2
Number of hash loops                    = 0
Number of hash join overflows           = 0
Number of small hash join overflows     = 0


Memory usage for database:

    Memory Pool Type                        = Backup/Restore/Util Heap
       Current size (bytes)                 = 65536
       High water mark (bytes)              = 65536
       Configured size (bytes)              = 20512768


    Memory Pool Type                        = Package Cache Heap
       Current size (bytes)                 = 720896
       High water mark (bytes)              = 720896
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Catalog Cache Heap
       Current size (bytes)                 = 65536
       High water mark (bytes)              = 65536
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Buffer Pool Heap
       Current size (bytes)                 = 4325376
       High water mark (bytes)              = 4325376
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Buffer Pool Heap
       Current size (bytes)                 = 589824
       High water mark (bytes)              = 589824
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Buffer Pool Heap
       Current size (bytes)                 = 327680
       High water mark (bytes)              = 327680
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Buffer Pool Heap
       Current size (bytes)                 = 196608
       High water mark (bytes)              = 196608
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Buffer Pool Heap
       Current size (bytes)                 = 131072
       High water mark (bytes)              = 131072
       Configured size (bytes)              = 4294967296

    Memory Pool Type                        = Shared Sort Heap
       Current size (bytes)                 = 65536
       High water mark (bytes)              = 327680
       Configured size (bytes)              = 20512768
```

```
Memory Pool Type                        = Lock Manager Heap
   Current size (bytes)                 = 655360
   High water mark (bytes)              = 655360
   Configured size (bytes)              = 720896

Memory Pool Type                        = Database Heap
   Current size (bytes)                 = 4653056
   High water mark (bytes)              = 4653056
   Configured size (bytes)              = 8978432

Memory Pool Type                        = Other Memory
   Current size (bytes)                 = 196608
   High water mark (bytes)              = 196608
   Configured size (bytes)              = 4294967296
```

```
4. Figure 4
Change the following
--#SET PERF_DETAIL 1 ROWS_OUT 5
Query 2
Statement number: 2
select lastname, firstnme,
deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
employee.empno = emp_act.empno and
emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2
LASTNAME         FIRSTNME      DEPTNAME                       NUM_ACT
------------------------------------------------------------------------
GEYER            JOHN          SUPPORT SERVICES                     2
GOUNOT           JASON         SOFTWARE SUPPORT                     2
HAAS             CHRISTINE     SPIFFY COMPUTER SERVICE DIV.         2
JONES            WILLIAM       MANUFACTURING SYSTEMS                2
KWAN             SALLY         INFORMATION CENTER                   2
Number of rows retrieved is:        8
Number of rows sent to output is:   5
Elapsed Time is:           0.037      seconds
Summary of Results
==================
                Elapsed             Agent CPU         Rows      Rows
Statement #     Time (s)            Time (s)          Fetched   Printed
1                    0.074                0.020         5         5
2                    0.037       Not Collected         8         5
Arith. mean     0.055
Geom.  mean     0.052
to

* Comment: " Query 2"
-----------------------------------------------

* SQL Statement Number 2:

select lastname, firstnme,
       deptname, count(*) as num_act
from employee, department, emp_act
where employee.workdept = department.deptno and
      employee.empno = emp_act.empno and
      emp_act.emptime < 1
group by lastname, firstnme, deptname
having count(*) <= 2;

LASTNAME        FIRSTNME     DEPTNAME                       NUM_ACT
--------------- ------------ ------------------------------ -----------
GEYER           JOHN         SUPPORT SERVICES                         2
GOUNOT          JASON        SOFTWARE SUPPORT                         2
HAAS            CHRISTINE    SPIFFY COMPUTER SERVICE DIV.             2
JONES           WILLIAM      MANUFACTURING SYSTEMS                    2
KWAN            SALLY        INFORMATION CENTER                       2
* 8 row(s) fetched, 5 row(s) output.

* Elapsed Time is:       0.004534 seconds

* Summary Table:

Type      Number      Repetitions Total Time (s) Min Time (s)   ...
--------- ----------- ----------- -------------- -------------- 
Statement           1           1       0.046319       0.046319 ...
Statement           2           1       0.004534       0.004534 ...

...Max Time (s)   Arithmetic Mean Geometric Mean Row(s) Fetched Row(s) Output
   -------------- --------------- -------------- -------------- -------------
...      0.046319        0.046319       0.046319              5             5
...      0.004534        0.004534       0.004534              8             5


* Total Entries:              2
* Total Time:                 0.050853 seconds
* Minimum Time:               0.004534 seconds
* Maximum Time:               0.046319 seconds
* Arithmetic Mean Time:       0.025426 seconds
* Geometric Mean Time:        0.014492 seconds
-----------------------------------------------
* Timestamp: Thu Feb 02 2006 16:03:18 EST
```


Figure 36. Sample Output from db2batch (Part 2)

The above sample output includes specific data elements returned by the database system monitor.

In the next example (on UNIX), just the summary table is produced.

```
db2batch -d sample -f db2batch.sql -r /dev/null,
```

Produces just the materialized query table. Using the `-r` option, `outfile1` was replaced by `/dev/null` and `outfile2` (which contains just the materialized query table) is empty, so `db2batch` sends the output to the screen:

```
* Summary Table:

Type      Number      Repetitions Total Time (s) Min Time (s)   ...
--------- ----------- ----------- -------------- -------------- 
Statement           1           1       0.052655       0.052655 ...
Statement           2           1       0.004518       0.004518 ...


...Max Time (s)   Arithmetic Mean Geometric Mean Row(s) Fetched Row(s) Output
   -------------- --------------- -------------- -------------- -------------
...      0.052655        0.052655       0.052655              5             5
...      0.004518        0.004518       0.004518              8             5


* Total Entries:             2
* Total Time:                0.057173 seconds
* Minimum Time:              0.004518 seconds
* Maximum Time:              0.052655 seconds
* Arithmetic Mean Time:      0.028587 seconds
* Geometric Mean Time:       0.015424 seconds
```

Figure 37. Sample Output from db2batch -- Summary Table Only

Related concepts:

- "Benchmark test analysis example" on page 373
- "Benchmark test creation" on page 359
- "Benchmark test execution" on page 371

Benchmark test execution

For one type of database benchmark, you choose a configuration parameter and run the test with different values for that parameter until the maximum benefit is achieved. A single test should include executing the application through several iterations (for example, 20 or 30 times) with the same parameter value to get an average timing, which shows the effect of parameter changes more clearly.

When you run the benchmark, the first iteration, which is called a warm-up run, should be considered a separate case from the subsequent iterations, which are called normal runs. Because the warm-up run includes some start-up activities, such as initializing the buffer pool, and consequently, takes somewhat longer than normal runs. Although the information from the warm-up run might be realistically valid, it is not statistically valid. When you calculate the average timing or CPU for a specific set of parameter values, use only the results from normal runs.

You might consider using the Configuration Advisor to create the warm-up run of the benchmark. The questions that the Configuration Advisor asks can provide insight into some things to consider when you adjust the configuration of your environment for the normal runs during your benchmark activity. You can start the

Configuration Advisor from the Control Center or by executing the db2 autoconfigure command with appropriate options.

If benchmarking uses individual queries, ensure that you minimize the potential effects of previous queries by flushing the buffer pool. To flush the buffer pool, read a number of pages that irrelevant to your query and to fill the buffer pool.

After you complete the iterations for a single set of parameter values, you can change a single parameter. However, between each iteration, perform the following tasks to restore the benchmark environment to its original state:

- . If the catalog statistics were updated for the test, make sure that the same values for the statistics are used for every iteration.
- The data used in the tests must be consistent if it is updated by the tests. This can be done by:
  - Using the RESTORE utility to restore the entire database. The backup copy of the database contains its previous state, ready for the next test.
  - Using the IMPORT or LOAD utility to restore an exported copy of the data. This method allows you to restore only the data that has been affected. REORG and RUNSTATS utilities should be run against the tables and indexes that contain this data.
- To return the application to its original state, re-bind it to the database.

In summary, follow these steps or iterations to benchmark a database application:

Step 1 Leave the database and database manager tuning parameters at their **default** values except for:
- Those parameters significant to the workload and the objectives of the test. (You rarely have enough time to perform benchmark testing to tune all of the parameters, so you may want to start by using your best guess for some of the parameters and tune from that point.)
- Log sizes, which should be determined during unit and system testing of your application.
- Any parameters that must be changed to enable your application to run (that is, the changes needed to prevent negative SQL return codes from such events as running out of memory for the statement heap).

Run your set of iterations for this initial case and calculate the average timing or CPU.

Step 2 Select one and only one tuning parameter to be tested, and change its value.

Step 3 Run another set of iterations and calculate the average timing or CPU.

Step 4 Depending on the results of the benchmark test, do one of the following:
- If performance improves, change the value of the same parameter and return to Step 3. Keep changing this parameter until the maximum benefit is shown.
- If performance degrades or remains unchanged, return the parameter to its previous value, return to Step 2, and select a new parameter. Repeat this procedure until all parameters have been tested.

  Note: If you were to graph the performance results, you would be looking for the point where the curve begins to plateau or decline.

You can write a driver program to help you with your benchmark testing. This driver program could be written using a language such as REXX or, for UNIX-based platforms, using shell scripts.

This driver program would execute the benchmark program, pass it the appropriate parameters, drive the test through multiple iterations, restore the environment to a consistent state, set up the next test with new parameter values, and collect/consolidate the test results. These driver programs can be flexible enough that they could be used to run the entire set of benchmark tests, analyze the results, and provide a report of the final and best parameter values for the given test.

Related concepts:

- "Benchmark preparation" on page 358
- "Benchmark testing" on page 357
- "Examples of db2batch tests" on page 360

Benchmark test analysis example

Output from the benchmark program should include an identifier for each test, the iteration of the program execution, the statement number, and the timing for the execution.

A summary of benchmarking results after a series of measurements might look like the following:

```
Test     Iter.   Stmt    Timing         SQL Statement
Numbr    Numbr   Numbr   (hh:mm:ss.ss)
 002      05      01     00:00:01.34    CONNECT TO SAMPLE
 002      05      10     00:02:08.15    OPEN cursor_01
 002      05      15     00:00:00.24    FETCH cursor_01
 002      05      15     00:00:00.23    FETCH cursor_01
 002      05      15     00:00:00.28    FETCH cursor_01
 002      05      15     00:00:00.21    FETCH cursor_01
 002      05      15     00:00:00.20    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      15     00:00:00.22    FETCH cursor_01
 002      05      20     00:00:00.84    CLOSE cursor_01
 002      05      99     00:00:00.03    CONNECT RESET
```

Figure 38. Benchmark Sample Results

Note: The data in the above report is shown for illustration purposes only. It does **not** represent measured results.

Analysis shows that the CONNECT (statement 01) took 1.34 seconds, the OPEN CURSOR (statement 10) took 2 minutes and 8.15 seconds, the FETCHES (statement 15) returned seven rows with the longest delay being .28 seconds, the CLOSE CURSOR (statement 20) took .84 seconds, and the CONNECT RESET (statement 99) took .03 seconds.

If your program can output data in a delimited ASCII format, it could later be imported into a database table or a spreadsheet for further statistical analysis.

Sample output for a benchmark report might be:

```
PARAMETER        VALUES FOR EACH BENCHMARK TEST
TEST NUMBER      001     002     003    004     005
locklist         63      63      63      63      63
maxappls         8       8       8       8       8
applheapsz       48      48      48      48      48
dbheap           128     128     128     128     128
sortheap         256     256     256     256     256
maxlocks         22      22      22      22      22
stmtheap         1024    1024    1024    1024    1024
SQL STMT         AVERAGE TIMINGS (seconds)
  01             01.34   01.34   01.35   01.35   01.36
  10             02.15   02.00   01.55   01.24   01.00
  15             00.22   00.22   00.22   00.22   00.22
  20             00.84   00.84   00.84   00.84   00.84
  99             00.03   00.03   00.03   00.03   00.03
```

Figure 39. Benchmark Sample Timings Report

Note: The data in the above report is shown for illustration purposes only. It does **not** represent any measured results.

Related concepts:

- "Benchmark test creation" on page 359
- "Benchmark test execution" on page 371
- "Benchmark testing" on page 357

Part 4. Appendixes

Appendix A. DB2 Configuration Parameters

Parameter details by function

The following sections provide additional details to assist in understanding and tuning the different configuration parameters. This discussion of the individual parameters is organized based on their function or purpose:

The discussion of each parameter includes the following information:

Configuration Type
Indicates which configuration file contains the setting for the parameter:

- *Database manager* affects an instance of the database manager and all databases defined within that instance
- *Database* affects a specific database

Parameter Type
Indicates whether or not you can change the parameter value, and whether the change will take effect online:

- *Configurable*

  A range of values are possible and the parameter may need to be tuned based on the database administrator's knowledge of the applications or from benchmarking experience.
- *Configurable online*

  A range of values are possible and the parameter may need to be tuned based on the database administrator's knowledge of the applications and/or from benchmarking experience. Changes can be applied while the database is online, without having to stop and restart the database manager, or reactivate the database.
- *Informational*

  These parameters are changed only by the database manager itself and will contain information such as the release of DB2 that a database was created under or an indication that a required backup is pending.

Capacity management

There are a number of configuration parameters at both the database and database manager levels that can impact the throughput on your system. These parameters are categorized in the following groups:

- "Database shared memory"
- "Application shared memory" on page 392
- "Agent private memory" on page 395
- "Agent/application communication memory" on page 403
- "Database manager instance memory" on page 407
- "Locks" on page 412
- "I/O and storage" on page 416
- "Agents" on page 422
- "Stored procedures and user-defined functions" on page 432

Database shared memory

The following parameters affect the database global memory allocated on your system:

- "catalogcache_sz - Catalog cache size "
- "database_memory - Database shared memory size " on page 380
- "dbheap - Database heap " on page 382
- "db_mem_thresh - Database memory threshold " on page 381
- "locklist - Maximum storage for lock list " on page 383
- "logbufsz - Log buffer size " on page 386
- "pckcachesz - Package cache size " on page 387
- "self_tuning_mem- Self tuning memory " on page 389
- "sheapthres_shr - Sort heap threshold for shared sorts " on page 390
- "util_heap_sz - Utility heap size " on page 392

catalogcache_sz - Catalog cache size

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	-1 [8 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is initialized
When Freed	When the database is shut down

This parameter is allocated out of the database shared memory, and is used to cache system catalog information. In a partitioned database system, there is one catalog cache for each database partition.

Caching catalog information at individual database partitions allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs (or the catalog node in a partitioned database environment) to obtain information that has previously been retrieved. The catalog cache is used to store:

- SYSTABLES information (including packed descriptors)
- authorization information, including SYSDBAUTH information and execute privileges for routines
- SYSROUTINES information

The use of the catalog cache can help improve the overall performance of:

- binding packages and compiling SQL and XQuery statements
- operations that involve checking database-level privileges
- operations that involve checking execute privileges for routines
- applications that are connected to non-catalog nodes in a partitioned database environment

By taking the default (-1) in a server or partitioned database environment, the value used to calculate the page allocation is four times the value specified for the *maxappls* configuration parameter. The exception to this occurs if four times *maxappls* is less than 8. In this situation, the default value of -1 will set *catalogcache_sz* to 8.

Recommendation: Start with the default value and tune it by using the database system monitor. When tuning this parameter, you should consider whether the extra memory being reserved for the catalog cache might be more effective if it was allocated for another purpose, such as the buffer pool or package cache.

Tuning this parameter is particularly important if a workload involves many SQL or XQuery compilations for a brief period of time, with few or no compilations thereafter. If the cache is too large, memory might be wasted holding copies of information that will no longer be used.

In an partitioned database environment, consider if the *catalogcache_sz* at the catalog node needs to be set larger since catalog information that is required at non-catalog nodes will always first be cached at the catalog node.

The *cat_cache_lookups* (catalog cache lookups), *cat_cache_inserts* (catalog cache inserts), *cat_cache_overflows* (catalog cache overflows), and *cat_cache_size_top* (catalog cache high water mark) monitor elements can help you determine whether you should adjust this configuration parameter.

Note: The catalog cache exists on all nodes in a partitioned database environment. Since there is a local database configuration file for each node, each node's *catalogcache_sz* value defines the size of the local catalog cache. In order to provide efficient caching and avoid overflow scenarios, you need to explicitly set the *catalogcache_sz* value at each node and consider the feasibility of possibly setting the *catalogcache_sz* on non-catalog nodes to be smaller than that of the catalog node; keep in mind that information that is required to be cached at non-catalog nodes will be retrieved from the catalog node's cache. Hence, a catalog cache at a non-catalog node is like a subset of the information in the catalog cache at the catalog node.

In general, more cache space is required if a unit of work contains several dynamic SQL or XQuery statements or if you are binding packages that contain a large number of static SQL or XQuery statements.

Related reference:

- "maxappls - Maximum number of active applications " on page 427

- "cat_cache_size_top - Catalog Cache High Water Mark monitor element" in *System Monitor Guide and Reference*
- "cat_cache_inserts - Catalog Cache Inserts monitor element" in *System Monitor Guide and Reference*
- "cat_cache_lookups - Catalog Cache Lookups monitor element" in *System Monitor Guide and Reference*
- "cat_cache_overflows - Catalog Cache Overflows monitor element" in *System Monitor Guide and Reference*
- "Restrictions on native XML data store" in *XML Guide*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

database_memory - Database shared memory size

Configuration Type Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

AIX and Windows
Automatic [Automatic, Computed, 0 — 4 294 967 295]

Linux, HP-UX, Solaris Operating System
Computed [Computed, 0 — 4 294 967 295]

Unit of Measure Pages (4 KB)

When Allocated When the database is activated

When Freed When the database is deactivated

This parameter specifies the amount of shared memory that is reserved for the database shared memory region. If this amount is less than the amount calculated from the individual memory parameters (for example, locklist, utility heap, bufferpools, and so on), the larger amount will be used.

Setting this parameter to AUTOMATIC on Windows and AIX enables self tuning. When enabled, the memory tuner determines the overall memory requirements for the database and increases or decreases the amount of memory allocated for database shared memory depending on the current database requirements. For example, if the current database requirements are high, and there is sufficient free memory on the system, more memory will be consumed by database shared memory. Once the database memory requirements drop, or the amount of free memory on the system drops too low, some database shared memory is released.

The memory tuner will always leave a minimum amount of memory free based on the calculated benefit to providing additional memory to the instance. If there is a great benefit to providing an instance with more memory, then the memory tuner

will maintain a lower amount of free memory. If the benefit is lower, then more free memory will be maintained. This allows databases to cooperate in the distribution of system memory.

Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning for self tuning to be active.

To simplify the management of this parameter, the COMPUTED setting instructs DB2 to calculate the amount of memory needed, and to allocate it at database activation time. DB2 will also allocate some additional memory to satisfy peak memory requirements for any heap in the database shared memory region whenever a heap exceeds its configured size. Other operations, such as dynamic configuration updates, also have access to this additional memory. The **db2pd** command, with the -memsets option, can be used to monitor the amount of unused memory left in the database shared memory region. On 64-bit DB2 for AIX, the database real shared memory usage grows dynamically to accommodate the needs of the database up to the amount of physical memory on the system, so there is no need to explicitly control the *database_memory* parameter. However, setting the DB2_PINNED_BP or DB2_LGPAGE_BP registry variable will restrict the ability to grow database shared memory. See the description of these registry variables in "Performance variables".

Note: In DB2 Version 9.1, the COMPUTED setting is equivalent to the DB2 UDB Version 8 AUTOMATIC setting. In DB2 Version 9.1, the AUTOMATIC setting enables self tuning.

Recommendation: This value will usually remain at AUTOMATIC (AIX and Windows) or COMPUTED (Linux, HP-UX, or Solaris operating environment). For platforms that do not support the AUTOMATIC setting, this parameter can be used to reserve additional memory for future expansion by setting it to a value larger than the sum of the memory heaps contained in the database shared memory. For example, the additional memory can be used for creating new buffer pools, or for increasing the size of existing buffer pools.

Related concepts:

- "Self tuning memory" on page 255
- "What's new for V9.1: database_memory configuration parameter change" in *What's New*

Related reference:

- "db_mem_thresh - Database memory threshold " on page 381
- "self_tuning_mem- Self tuning memory " on page 389
- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

db_mem_thresh - Database memory threshold

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]	10 [0–100]
Unit of Measure	Percentage

This database configuration parameter relates to how the database manager handles excess unused database shared memory. Typically, as pages of memory are touched by a process, they are committed, meaning that a page of memory has been allocated by the operating system and occupies space either in physical memory or in a page file on disk. Depending on the database workload, there may be peak database shared memory requirements at a certain times of day. Once the operating system has enough committed memory to meet those peak requirements, that memory remains committed, even after peak memory requirements have subsided. This database configuration parameter represents the maximum percentage of committed, but currently unused, database shared memory that the database manager will allow before starting to release committed pages of memory back to the operating system. Acceptable values are whole numbers in the range of 0 (immediately release any unused database shared memory) to 100 (never release any unused database shared memory). The default is 10 (release unused memory only when more than 10% of database shared memory is currently unused).

The default value should be suitable for most workloads. Care should be taken when updating this parameter, as setting the value too low could cause excessive memory thrashing on the box (memory pages constantly being committed and then released), and setting the value too high might prevent the database manager from returning any database shared memory back to the operating system for other processes to use.

This configuration parameter can be updated dynamically.

This configuration parameter will be ignored (meaning that unused database shared memory pages will remain committed) if the database shared memory region is pinned through the DB2_PINNED_BP registry variable, configured for large pages through the DB2_LARGE_PAGE_MEM registry variable, or if releasing of memory is explicitly disabled through the DB2MEMDISCLAIM registry variable.

Some versions of Linux do not support releasing subranges of a shared memory segment back to the operating system. On such platforms, this parameter will be ignored.

Related reference:

- "database_memory - Database shared memory size " on page 380
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

dbheap - Database heap

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	
	UNIX 1200 [32 – 524 288]
	Windows Database server with local and remote clients 600 [32 – 524 288]

	Windows 64-bit Database server with local clients 600 [32 – 524 288]
	Windows 32-bit Database server with local clients 300 [32 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is activated
When Freed	When the database is deactivated

There is one database heap per database, and the database manager uses it on behalf of all applications connected to the database. It contains control block information for tables, indexes, table spaces, and buffer pools. It also contains space for the log buffer (*logbufsz*) and temporary memory used by utilities. Therefore, the size of the heap will be dependent on a large number of variables. The control block information is kept in the heap until all applications disconnect from the database.

The minimum amount the database manager needs to get started is allocated at the first connection. The data area is expanded as needed until all unreserved memory in the database shared memory region is consumed.

You can use the database system monitor to track the highest amount of memory that was used for the database heap, using the db_heap_top (maximum database heap allocated) element.

Related reference:

- "db_heap_top - Maximum Database Heap Allocated monitor element" in *System Monitor Guide and Reference*
- "logbufsz - Log buffer size " on page 386
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

locklist - Maximum storage for lock list

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	
	UNIX Automatic [4 – 524 288]
	Windows Database server with local and remote clients Automatic [4 – 524 288]
	Windows 64-bit Database server with local clients Automatic [4 – 524 288]
	Windows 32-bit Database server with local clients Automatic [4 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the first application connects to the database

When Freed When last application disconnects from the database

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database. Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked. The database manager can also acquire locks for internal use.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active

The value of *locklist* is tuned together with the *maxlocks* parameter, therefore disabling self tuning of the *locklist* parameter automatically disables self tuning of the *maxlocks* parameter. Enabling self tuning of the *locklist* parameter automatically enables self tuning of the *maxlocks* parameter.

On 32-bit platforms, each lock requires 48 or 96 bytes of the lock list, depending on whether other locks are held on the object:

- 96 bytes are required to hold a lock on an object that has no other locks held on it
- 48 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit platforms (except HP-UX/PA-RISC), each lock requires 64 or 128 bytes of the lock list, depending on whether other locks are held on the object:

- 128 bytes are required to hold a lock on an object that has no other locks held on it
- 64 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit HP-UX/PA-RISC, each lock requires 80 or 160 bytes of the lock list, depending on whether or not other locks are held on the object.

When the percentage of the lock list used by one application reaches *maxlocks*, the database manager will perform lock escalation, from row to table, for the locks held by the application. Although the escalation process itself does not take much time, locking entire tables (versus individual rows) decreases concurrency, and overall database performance might decrease for subsequent accesses against the affected tables. Suggestions of how to control the size of the lock list are:

- Perform frequent COMMITs to release locks.
- When performing many updates, lock the entire table before updating (using the SQL LOCK TABLE statement). This will use only one lock, keeps others from interfering with the updates, but does reduce concurrency of the data.

  You can also use the LOCKSIZE option of the ALTER TABLE statement to control how locking is done for a specific table.

  Use of the Repeatable Read isolation level might result in an automatic table lock.

- Use the Cursor Stability isolation level when possible to decrease the number of share locks held. If application integrity requirements are not compromised use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.
- Set *locklist* to AUTOMATIC. The lock list will increase synchronously to avoid lock escalation or a lock list full situation.

Once the lock list is full, performance can degrade since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Additionally there might be more deadlocks between applications (since they are all waiting on a limited number of table locks), which will result in transactions being rolled back. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

Recommendation: If lock escalations are causing performance concerns you might need to increase the value of this parameter or the *maxlocks* parameter. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *lock_escals (lock escalations)* monitor element.

The following steps might help in determining the number of pages required for your lock list:

1. Calculate a lower bound for the size of your lock list, using *one* of the following calculations, depending on your environment:
   a. `(512 * x * maxappls) / 4096`
   b. with Concentrator enabled:
      `(512 * x * max_coordagents) / 4096`
   c. in a partitioned database with Concentrator enabled:
      `(512 * x * max_coordagents * number of database partitions) / 4096`

   where 512 is an estimate of the average number of locks per application and `x` is the number of bytes required for each lock against an object that has an existing lock (40 bytes on 32-bit platforms, 64 bytes on 64-bit platforms).
2. Calculate an upper bound for the size of your lock list:
   `(512 * y * maxappls) / 4096`

   where `y` is the number of bytes required for the first lock against an object (80 bytes on 32-bit platforms, 128 bytes on 64-bit platforms).
3. Estimate the amount of concurrency you will have against your data and based on your expectations, choose an initial value for *locklist* that falls between the upper and lower bounds that you have calculated.
4. Using the database system monitor, as described below, tune the value of this parameter.

You can use the database system monitor to determine the maximum number of locks held by a given transaction. Refer to the *locks_held_top (maximum number of locks held)* monitor element.

This information can help you validate or adjust the estimated number of locks per application. In order to perform this validation, you will have to sample several applications, noting that the monitor information is provided at a transaction level, not an application level.

You might also want to increase *locklist* if *maxappls* is increased, or if the applications being run perform infrequent commits.

You should consider rebinding applications (using the REBIND command) after changing this parameter.

Related concepts:

- "Self tuning memory" on page 255

Related reference:

- "locks_held_top - Maximum Number of Locks Held monitor element" in *System Monitor Guide and Reference*
- "lock_escals - Number of Lock Escalations monitor element" in *System Monitor Guide and Reference*
- "REBIND command" in *Command Reference*
- "maxappls - Maximum number of active applications " on page 427
- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "self_tuning_mem- Self tuning memory " on page 389
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logbufsz - Log buffer size

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	
	32-bit platforms 8 [4 — 4 096]
	64-bit platforms 8 [4 — 65 535]
Unit of Measure	Pages (4 KB)

This parameter allows you to specify the amount of the database heap (defined by the *dbheap* parameter) to use as a buffer for log records before writing these records to disk. The log records are written to disk when one of the following occurs:

- A transaction commits or a group of transactions commit, as defined by the *mincommit* configuration parameter
- The log buffer is full
- As a result of some other internal database manager event.

This parameter must also be less than or equal to the *dbheap* parameter. Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently and more log records will be written at each time.

Recommendation: Increase the size of this buffer area if there is considerable read activity on a dedicated log disk, or there is high disk utilization. When increasing the value of this parameter, you should also consider the *dbheap* parameter since the log buffer area uses space controlled by the *dbheap* parameter.

You can use the database system monitor to determine how much of the log buffer space is used for a particular transaction (or unit of work). Refer to the *log_space_used* (unit of work log space used) monitor element.

Related reference:

- "uow_log_space_used - Unit of Work Log Space Used monitor element" in *System Monitor Guide and Reference*
- "mincommit - Number of commits to group " on page 449
- "dbheap - Database heap " on page 382
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

pckcachesz - Package cache size

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	
	32-bit platforms Automatic [-1, 32 — 128 000]
	64-bit platforms Automatic [-1, 32 — 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the database is initialized
When Freed	When the database is shut down

This parameter is allocated out of the database shared memory, and is used for caching of sections for static and dynamic SQL and XQuery statements on a database. In a partitioned database system, there is one package cache for each database partition.

Caching packages allows the database manager to reduce its internal overhead by eliminating the need to access the system catalogs when reloading a package; or, in the case of dynamic SQL or XQuery statements, eliminating the need for compilation. Sections are kept in the package cache until one of the following occurs:

- The database is shut down
- The package or dynamic SQL or XQuery statement is invalidated
- The cache runs out of space.

This caching of the section for a static or dynamic SQL or XQuery statement can improve performance, especially when the same statement is used multiple times by applications connected to a database. This is particularly important in a transaction processing application.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. When *self_tuning_mem* is set to ON, the memory tuner will dynamically size the memory area controlled by *pckcachesz* as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

When this parameter is set to -1, the value used to calculate the page allocation is eight times the value specified for the *maxappls* configuration parameter. The exception to this occurs if eight times *maxappls* is less than 32. In this situation, the default value of -1 will set *pckcachesz* to 32.

Recommendation: When tuning this parameter, you should consider whether the extra memory being reserved for the package cache might be more effective if it was allocated for another purpose, such as the buffer pool or catalog cache. For this reason, you should use benchmarking techniques when tuning this parameter.

Tuning this parameter is particularly important when several sections are used initially and then only a few are run repeatedly. If the cache is too large, memory is wasted holding copies of the initial sections.

The following monitor elements can help you determine whether you should adjust this configuration parameter:

- *pkg_cache_lookups* (package cache lookups)
- *pkg_cache_inserts* (package cache inserts)
- *pkg_cache_size_top* (package cache high water mark)
- *pkg_cache_num_overflows* (package cache overflows)

Note: The package cache is a working cache, so you cannot set this parameter to zero. There must be sufficient memory allocated in this cache to hold all sections of the SQL or XQuery statements currently being executed. If there is more space allocated than currently needed, then sections are cached. These sections can simply be executed the next time they are needed without having to load or compile them.

The limit specified by the *pckcachesz* parameter is a soft limit. This limit can be exceeded, if required, if memory is still available in the database shared set. You can use the *pkg_cache_size_top* monitor element to determine the largest that the package cache has grown, and the *pkg_cache_num_overflows* monitor element to determine how many times the limit specified by the *pckcachesz* parameter has been exceeded.

Related concepts:

- "Self tuning memory" on page 255

Related reference:

- "pkg_cache_size_top - Package Cache High Water Mark monitor element" in *System Monitor Guide and Reference*
- "pkg_cache_inserts - Package Cache Inserts monitor element" in *System Monitor Guide and Reference*
- "pkg_cache_lookups - Package Cache Lookups monitor element" in *System Monitor Guide and Reference*
- "pkg_cache_num_overflows - Package Cache Overflows monitor element" in *System Monitor Guide and Reference*
- "maxappls - Maximum number of active applications " on page 427
- "self_tuning_mem- Self tuning memory " on page 389
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

self_tuning_mem- Self tuning memory

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

Single-partition environments
ON [ON; OFF]

Multi-partition environments
OFF [ON; OFF]

When this parameter is set to ON, the memory tuner dynamically distributes available memory resources as required between memory consumers that are enabled for self tuning. Because memory is being traded between memory consumers, there must be at least two memory consumers enabled for self tuning in order for the memory tuner to be active. When *self_tuning_mem* is set to ON, but there are less than two memory consumers enabled for self tuning, the memory tuner is inactive. (The exception to this is the sort heap memory area, which can be tuned regardless of whether other memory consumers are enabled for self tuning or not.)

This parameter is ON by default in single database partition environments. In multi-partition environments, it is OFF by default.

The memory consumers that can be enabled for self tuning include:

- Buffer pools (controlled by the size parameter of the ALTER BUFFERPOOL and CREATE BUFFERPOOL statements
- Package cache (controlled by the *pckcachesz* configuration parameter)
- Lock List (controlled by the *locklist* and *maxlocks* configuration parameters)
- Sort heap (controlled by the *sheapthres_shr* and *sortheap* configuration parameters)
- Database shared memory (controlled by the *database_memory* configuration parameter)

To view the current setting for this parameter, use the GET DATABASE CONFIGURATION command specifying the SHOW DETAIL parameter. The possible settings returned for this parameter are:

```
Self Tuning Memory                          (SELF_TUNING_MEM) = OFF
Self Tuning Memory                          (SELF_TUNING_MEM) = ON (Active)
Self Tuning Memory                          (SELF_TUNING_MEM) = ON (Inactive)
```

If the parameter shows `ON (Active)` it means that the memory tuner is actively tuning the memory on the system. If the parameter shows `ON (Inactive)` it means that although the parameter is set ON, self-tuning is not occurring because there are less than two memory consumers enabled for self-tuning.

In partitioned database environments, the *self_tuning_mem* configuration parameter will only show ON (Active) for the database partition on which the tuner is running. On all other nodes SELF_TUNING_MEM will show ON (Inactive). As a result, to determine if memory tuner is active in a partitioned database, you must check the SELF_TUNING_MEM parameter on all database partitions.

If you have migrated to DB2 Version 9 from an earlier version of DB2 and you plan to use the self tuning memory feature, you should configure the following health indicators to disable threshold or state checking:

- Shared Sort Memory Utilization - db.sort_shrmem_util
- Percentage of sorts that overflowed - db.spilled_sorts
- Long Term Shared Sort Memory Utilization - db.max_sort_shrmem_util
- Lock List Utilization - db.locklist_util
- Lock Escalation Rate - db.lock_escal_rate
- Package Cache Hit Ratio - db.pkgcache_hitratio

One of the objectives of the self tuning memory feature is to avoid having memory allocated to a memory consumer when it is not immediately required. Therefore, utilization of the memory allocated to a memory consumer may approach 100% before more memory is allocated. By disabling these health indicators, you will avoid unnecessary alerts triggered by the high rate of memory utilization by a memory consumer.

Instances created in DB2 Version 9 will have these health indicators disabled by default.

In a database that is migrated from an earlier version of DB2, *self_tuning_mem* will be set to OFF.

Related concepts:

- "Self tuning memory" on page 255

Related tasks:

- "Enabling self tuning memory" on page 256

Related reference:

- "ALTER BUFFERPOOL statement" in *SQL Reference, Volume 2*
- "database_memory - Database shared memory size " on page 380
- "locklist - Maximum storage for lock list " on page 383
- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "pckcachesz - Package cache size " on page 387
- "sheapthres - Sort heap threshold " on page 399
- "sheapthres_shr - Sort heap threshold for shared sorts " on page 390
- "sortheap - Sort heap size " on page 400

sheapthres_shr - Sort heap threshold for shared sorts

Configuration Type Database

Parameter Type Configurable online

Propagation class Immediate

Default [Range]

32-bit platforms
5000 [Automatic, 250 — 2 097 152]

64-bit platforms
5000 [Automatic, 250 — 2 147 483 647]

Unit of Measure Pages (4 KB)

This parameter represents a soft limit on the total amount of database shared memory that can be used by sort memory consumers at any one time. There are other sort memory consumers in addition to sort, like hash join, index anding, block index anding, merge join and in-memory tables.When the total amount of shared memory for shared sort memory consumers approaches the *sheapthres_shr* limit, a memory throttling mechanism is activated and the future shared sort memory consumer requests may be granted less memory than requested, but will always be granted more than the minimum they need for finishing the task. Once the *sheapthres_shr* limit is exceeded, all requests of shared sort memory from sort memory consumers will be granted the minimum amount of memory required to finish the task. When the total amount of shared memory for active shared sort memory consumers reaches this limit, subsequent sorts could fail (SQL0955C).

When the value of the database manager configuration parameter *sheapthres* is 0, all sort memory consumers for the database will use the database shared memory with *sheapthres_shr* instead of private sort memory.

When *sheapthres_shr* is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

Automatic tuning of *sheapthres_shr* is allowed only when the database manager configuration parameter *sheapthres* is set to 0.

The value of *sortheap* is tuned together with the *sheapthres_shr* parameter therefore disabling self tuning of the *sortheap* parameter automatically disables self tuning of the *sheapthres_shr* parameter. Enabling self tuning of the *sheapthres_shr* parameter automatically enables self tuning of the *sortheap* parameter.

When the value of this parameter is updated online, only new requests of shared-sort memory made after the update will use the new value. It is recommended that you reduce the value of *sortheap* before reducing the value of *sheapthres_shr* and to increase the value of *sheapthres_shr* before increasing the value of *sortheap*.

When the database manager configuration parameter *sheapthres* is greater than 0, *sheapthres_shr* is only meaningful in two cases:

- if the *intra_parallel* database manager configuration parameter is set to *yes*, because when *intra_parallel* is set to *no*, there will be no shared sorts.
- if the Concentrator is on (that is, when *max_connections* is greater than *max_coordagents*), because sorts that use a cursor declared with the WITH HOLD option will be allocated from shared memory.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "sheapthres - Sort heap threshold " on page 399

util_heap_sz - Utility heap size

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	5000 [16 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	As required by the database manager utilities
When Freed	When the utility no longer needs the memory

This parameter indicates the maximum amount of memory that can be used simultaneously by the BACKUP, RESTORE, and LOAD (including load recovery) utilities.

Recommendation: Use the default value unless your utilities run out of space, in which case you should increase this value. If memory on your system is constrained, you might wish to lower the value of this parameter to limit the memory used by the database utilities. If the parameter is set too low, you might not be able to run utilities concurrently. You should update this parameter dynamically as needed. For a small number of utilities, set this parameter to a small value. For a large number of utilites, or for memory intensive utilities, you should set this parameter to a larger value.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Application shared memory

The following parameters specify the work area that is used by all agents (both coordinating and subagents) that work for an application:

- "app_ctl_heap_sz - Application control heap size "
- "appgroup_mem_sz - Maximum size of application group memory set " on page 394
- "groupheap_ratio - Percent of memory for application group heap " on page 395

app_ctl_heap_sz - Application control heap size

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	

Database server with local and remote clients
128 [1–64 000] when INTRA_PARALLEL is not enabled

512 [1–64 000] when INTRA_PARALLEL is enabled

Database server with local clients
64 [1–64 000] (for non-UNIX platforms) when INTRA_PARALLEL is not enabled

	512 [1–64 000] (for non-UNIX platforms) when INTRA_PARALLEL is enabled 128 [1–64 000] (for UNIX-based platforms) when INTRA_PARALLEL is not enabled 512 [1–64 000] (for UNIX-based platforms) when INTRA_PARALLEL is enabled
	Partitioned database server with local and remote clients 512 [1–64 000]
Unit of Measure	Pages (4 KB)
When Allocated	When an application starts
When Freed	When an application completes

For partitioned databases, and for non-partitioned databases with intra-parallelism enabled (intra_parallel=ON), this parameter specifies the average size of the shared memory area allocated for an application. For non-partitioned databases where intra-parallelism is disabled (intra_parallel=OFF), this is the maximum private memory that will be allocated for the heap. There is one application control heap per connection per database partition.

The application control heap is required primarily for sharing information between agents working on behalf of the same request. Usage of this heap is minimal for non-partitioned databases when running queries with a degree of parallelism equal to 1.

This heap is also used to store descriptor information for declared temporary tables. The descriptor information for all declared temporary tables that have not been explicitly dropped is kept in this heap's memory and cannot be dropped until the declared temporary table is dropped.

Recommendation: Initially, start with the default value. You might have to set the value higher if you are running complex applications, if you have a system that contains a large number of database partitions, or if you use declared temporary tables. The amount of memory needed increases with the number of concurrently active declared temporary tables. A declared temporary table with many columns has a larger table descriptor size than a table with few columns, so having a large number of columns in an application's declared temporary tables also increases the demand on the application control heap.

Related reference:

- "intra_parallel - Enable intra-partition parallelism " on page 491
- "applheapsz - Application heap size " on page 397
- "appgroup_mem_sz - Maximum size of application group memory set " on page 394
- "groupheap_ratio - Percent of memory for application group heap " on page 395
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

appgroup_mem_sz - Maximum size of application group memory set

Configuration Type Database

Parameter Type Configurable

Default [Range]

UNIX Database server with local clients (other than 32-bit HP-UX)
20 000 [1 – 1 000 000]

32-bit HP-UX

- Database server with local clients
- Database server with local and remote clients
- Partitioned database server with local and remote clients

10 000 [1 – 1 000 000]

Windows Database server with local clients
10 000 [1 – 1 000 000]

Database server with local and remote clients (other than 32-bit HP-UX)
30 000 [1 – 1 000 000]

Partitioned database server with local and remote clients (other than 32-bit HP-UX)
40 000 [1 – 1 000 000]

Unit of Measure Pages (4 KB)

This parameter determines the size of the application group shared memory segment. Information that needs to be shared between agents working on the same application is stored in the application group shared memory segment.

In a partitioned database, or in a non-partitioned database with intra-partition parallelism enabled or concentrator enabled, multiple applications share one application group. One application group shared memory segment is allocated for the application group. Within the application group shared memory segment, each application will have its own application control heap, and all applications will share one application group shared heap.

The number of applications in one application group is calculated by:

```
appgroup_mem_sz / app_ctl_heap_sz
```

The application group shared heap size is calculated by:

```
appgroup_mem_sz * groupheap_ratio / 100
```

The size of each application control heap is calculated by:

```
app_ctl_heap_sz * (100 - groupheap_ratio) / 100
```

Recommendation: Retain the default value of this parameter unless you are experiencing performance problems.

Related reference:

- “app_ctl_heap_sz - Application control heap size ” on page 392
- “groupheap_ratio - Percent of memory for application group heap ” on page 395
- “GET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION command” in *Command Reference*

groupheap_ratio - Percent of memory for application group heap

Configuration Type Database

Parameter Type Configurable

Default [Range] 70 [1 – 99]

Unit of Measure Percentage

This parameter specifies the percentage of memory in the application control shared memory set devoted to the application group shared heap.

This parameter does not have any effect on a non-partitioned database with concentrator OFF and intra-partition parallelism disabled.

Recommendation: Retain the default value of this parameter unless you are experiencing performance problems.

Related reference:

- “app_ctl_heap_sz - Application control heap size ” on page 392
- “appgroup_mem_sz - Maximum size of application group memory set ” on page 394
- “GET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION command” in *Command Reference*

Agent private memory

The following parameters affect the amount of memory used for each database agent:

- “agent_stack_sz - Agent stack size ”
- “applheapsz - Application heap size ” on page 397
- “query_heap_sz - Query heap size ” on page 398
- “sheapthres - Sort heap threshold ” on page 399
- “sortheap - Sort heap size ” on page 400
- “stat_heap_sz - Statistics heap size ” on page 402
- “stmtheap - Statement heap size ” on page 402

agent_stack_sz - Agent stack size

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]	16 [8 – 1000]
Unit of Measure	Pages (4 KB)
When Allocated	When an agent is initialized to do work for an application
When Freed	When an agent completes the work to be done for an application

The agent stack is the virtual memory that is allocated by DB2 for each agent. This memory is committed when it is required to process an SQL or XQuery statement. You can use this parameter to optimize memory utilization of the server for a given set of applications. More complex queries will use more stack space, compared to the space used for simple queries.

This parameter is used to set the initial committed stack size for each agent in a Windows environment. By default, each agent stack can grow up to the default reserve stack size of 256 KB (64 4-KB pages). This limit is sufficient for most database operations.

However, when preparing a large SQL or XQuery statement, the agent can run out of stack space and the system will generate a stack overflow exception (0xC000000D). When this happens, the server will shut down because the error is non-recoverable.

The agent stack size can be increased by setting *agent_stack_sz* to a value larger than the default reserve stack size of 64 pages. Note that the value for *agent_stack_sz*, when larger than the default reserve stack size, is rounded by the Windows operating system to the nearest multiple of 1 MB; setting the agent stack size to 128 4-KB pages actually reserves a 1 MB stack for each agent. Setting the value for *agent_stack_sz* less than the default reserve stack size will have no effect on the maximum limit because the stack still grows if necessary up to the default reserve stack size. In this case, the value for *agent_stack_sz* is the initial committed memory for the stack when an agent is created.

You can change the default reserve stack size by using the db2hdr utility to change the header information for the `db2syscs.exe` file. Changing the default reserve stack size will affect all threads while changing *agent_stack_sz* only affects the stack size for agents. The advantage of changing the default stack size using the db2hdr utility is that it provides a better granularity, therefore allowing the stack size to be set at the minimum required stack size. However, you will have to stop and restart DB2 for a change to `db2syscs.exe` to take effect.

Recommendation: If you will be working with large or complex XML data in a 32-bit environment, you should update *agent_stack_sz* to at least 256 4-KB pages. Very complex XML schemas may require *agent_stack_sz* to be set much closer to the limit in order to avoid stack overflow exceptions during schema registration or during XML document validation.

You might be able to reduce the stack size in order to make more address space available to other clients, if your environment matches the following:

- Contains only simple applications (for example light OLTP), in which there are never complex queries
- Requires a relatively large number of concurrent clients (for example, more than 100).

The agent stack size and the number of concurrent clients are inversely related: a larger stack size reduces the potential number of concurrent clients that can be running. This occurs because address space is limited on Windows platforms.

This parameter does not apply to UNIX-based platforms.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

applheapsz - Application heap size

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	
	32-bit Database server with local and remote clients 256 [16 – 60 000]
	64-bit Database server with local and remote clients 256 [16 – 60 000]
	32-bit Partitioned database server with local and remote clients 64 [16 – 60 000]
	64-bit Partitioned database server with local and remote clients 128 [16 – 60 000]
Unit of Measure	Pages (4 KB)
When Allocated	When an agent is initialized to do work for an application
When Freed	When an agent completes the work to be done for an application

This parameter defines the number of private memory pages available to be used by the database manager on behalf of a specific agent or subagent.

The heap is allocated when an agent or subagent is initialized for an application. The amount allocated will be the minimum amount needed to process the request given to the agent or subagent. As the agent or subagent requires more heap space to process larger SQL or XQuery statements, the database manager will allocate memory as needed, up to the maximum specified by this parameter.

Notes:

1.

Note: In a partitioned database environment, the application control heap (*app_ctl_heap_sz*) is used to store copies of the executing sections of SQL statements for agents and subagents. SMP subagents, however, use *applheapsz*, as do agents in all other environments.

2.

Note: Large XML schemas require sufficient space in the application heap. It is recommended that you enlarge the application heap as follows:

```
db2 update db cfg using applheapsz 1000
```

Recommendation: Increase the value of this parameter if your applications receive an error indicating that there is not enough storage in the application heap.

The application heap (*applheapsz*) is allocated out of agent private memory.

Related reference:

- "app_ctl_heap_sz - Application control heap size " on page 392
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

query_heap_sz - Query heap size

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	1000 [2 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When an application (either local or remote) connects to the database
When Freed	When the application disconnects from the database, or detaches from the instance

This parameter specifies the **maximum** amount of memory that can be allocated for the query heap. A query heap is used to store each query in the agent's private memory. The information for each query consists of the input and output SQLDA, the statement text, the SQLCA, the package name, creator, section number, and consistency token. This parameter is provided to ensure that an application does not consume unnecessarily large amounts of virtual memory within an agent.

The query heap is also used for the memory allocated for blocking cursors. This memory consists of a cursor control block and a fully resolved output SQLDA.

The initial query heap allocated will be the same size as the application support layer heap, as specified by the *aslheapsz* parameter. The query heap size must be greater than or equal to two (2), and must be greater than or equal to the *aslheapsz* parameter. If this query heap is not large enough to handle a given request, it will be reallocated to the size required by the request (not exceeding *query_heap_sz*). If this new query heap is more than 1.5 times larger than *aslheapsz*, the query heap will be reallocated to the size of *aslheapsz* when the query ends.

Recommendation: In most cases the default value will be sufficient. As a minimum, you should set *query_heap_sz* to a value at least five times larger than *aslheapsz*. This will allow for queries larger than *aslheapsz* and provide additional memory for three or four blocking cursors to be open at a given time.

If you have very large LOBs, you might need to increase the value of this parameter so the query heap will be large enough to accommodate those LOBs.

Related reference:

- "aslheapsz - Application support layer heap size " on page 403
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

sheapthres - Sort heap threshold

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable online

Propagation class Immediate

Default [Range]

UNIX 32-bit platforms
0 [0 — 2 097 152]

Windows 32-bit platforms
0 [0 — 2 097 152]

64-bit platforms
0 [0 — 2 147 483 647]

Unit of Measure Pages (4 KB)

This parameter is an instance-wide soft limit on the total amount of memory that can be consumed by private sorts at any given time. When the total private sort memory consumption for an instance reaches this limit, the memory allocated for additional incoming private sort requests is considerably reduced.

Examples of operations that use the sort heap include: sorts, hash joins, dynamic bitmaps (used for index ANDing and Star Joins), and table in-memory operations.

Explicit definition of the threshold prevents the database manager from using excessive amounts of memory for large numbers of sorts.

There is no reason to increase the value of this parameter when moving from a non-partitioned to a partitioned database environment. Once you have tuned the database and database manager configuration parameters on a single database partition environment, the same values will in most cases work well in a

partitioned database environment. The only way to set this parameter to different values on different nodes or database partitions is to create more than one DB2 instance. This will require managing different DB2 databases over different database partition groups. Such an arrangement defeats the purpose of many of the advantages of a partitioned database environment.

When the instance-level *sheapthres* is set to 0, then the tracking of sort memory consumption is done at the database level only and memory allocation for sorts is constrained by the value of the database-level *sheapthres_shr* configuration parameter.

Automatic tuning of *sheapthres_shr* is allowed only when the database manager configuration parameter *sheapthres* is set to 0.

This parameter will not be dynamically updatable if any of the following are true:

- The starting value for *sheapthres* is 0 and the target value is a value different from 0.
- The starting value for *sheapthres* is a value different from 0 and the target value is 0.

Recommendation: Ideally, you should set this parameter to a reasonable multiple of the largest *sortheap* parameter you have in your database manager instance. This parameter should be **at least** two times the largest *sortheap* defined for any database within the instance.

If you are doing private sorts and your system is not memory constrained, an ideal value for this parameter can be calculated using the following steps:

1. Calculate the typical sort heap usage for each database:

```
    (typical number of concurrent agents running against the database)
  * (sortheap, as defined for that database)
```

2. Calculate the sum of the above results, which provides the total sort heap that could be used under typical circumstances for all databases within the instance.

You should use benchmarking techniques to tune this parameter to find the proper balance between sort performance and memory usage.

You can use the database system monitor to track the sort activity, using the post threshold sorts (*post_threshold_sorts*) monitor element.

Related reference:

- "post_threshold_sorts - Post Threshold Sorts monitor element" in *System Monitor Guide and Reference*
- "sheapthres_shr - Sort heap threshold for shared sorts " on page 390
- "sortheap - Sort heap size " on page 400
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

sortheap - Sort heap size

Configuration Type Database

Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	
	32-bit platforms Automatic [16 – 524 288]
	64-bit platforms Automatic [16 – 4 194 303]
Unit of Measure	Pages (4 KB)
When Allocated	As needed to perform sorts
When Freed	When sorting is complete

This parameter defines the maximum number of private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts. If the sort is a private sort, then this parameter affects agent private memory. If the sort is a shared sort, then this parameter affects the database shared memory. Each sort has a separate sort heap that is allocated as needed, by the database manager. This sort heap is the area where data is sorted. If directed by the optimizer, a smaller sort heap than the one specified by this parameter is allocated using information provided by the optimizer.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change.

The value of *sortheap* is tuned together with the *sheapthres_shr* parameter, therefore disabling self tuning of the *sortheap* parameter automatically disables self tuning of the *sheapthres_shr* parameter. Enabling self tuning of the *sheapthres_shr* parameter automatically enables self tuning of the *sortheap* parameter. The *sortheap* parameter can, however, be enabled for self tuning without the *sheapthres_shr* parameter being AUTOMATIC.

Automatic tuning of *sortheap* is allowed only when the database manager configuration parameter *sheapthres* is set to 0.

Recommendation: When working with the sort heap, you should consider the following:

- Appropriate indexes can minimize the use of the sort heap.
- Hash join buffers, block index ANDing, merge join, table in memory and dynamic bitmaps (used for index ANDing and Star Joins) use sort heap memory. Increase the size of this parameter when these techniques are used.
- Increase the size of this parameter when frequent large sorts are required.
- When increasing the value of this parameter, you should examine whether the *sheapthres* and *sheapthres_shr* parameters in the database manager configuration file also need to be adjusted.
- The sort heap size is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND command) after changing this parameter.

Related concepts:

- "Self tuning memory" on page 255

Related reference:

- "REBIND command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "self_tuning_mem- Self tuning memory " on page 389
- "sheapthres - Sort heap threshold " on page 399
- "sheapthres_shr - Sort heap threshold for shared sorts " on page 390

stat_heap_sz - Statistics heap size

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	4384 [1096 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the RUNSTATS utility is started
When Freed	When the RUNSTATS utility is completed

This parameter indicates the **maximum** size of the heap used in collecting statistics using the RUNSTATS command.

Recommendation: The default value is appropriate when no distribution statistics are collected or when distribution statistics are only being collected for relatively narrow tables. The minimum value is **not** recommended when distribution statistics are being gathered, as only tables containing 1 or 2 columns will fit in the heap.

You should adjust this parameter based on the number of columns for which statistics are being collected. Narrow tables, with relatively few columns, require less memory for distribution statistics to be gathered. Wide tables, with many columns, require significantly more memory. If you are gathering distribution statistics for tables which are very wide and require a large statistics heap, you might wish to collect the statistics during a period of low system activity so you do not interfere with the memory requirements of other users.

Related reference:

- "num_freqvalues - Number of frequent values retained " on page 479
- "num_quantiles - Number of quantiles for columns " on page 480
- "RUNSTATS command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

stmtheap - Statement heap size

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	

32-bit platforms
2048 [128 – 524 288]

64-bit platforms
4096 [128 – 524 288]

Unit of Measure Pages (4 KB)

When Allocated For each statement during precompiling or binding

When Freed When precompiling or binding of each statement is complete

The statement heap is used as a work space for the SQL or XQuery compiler during compilation of an SQL or XQuery statement. This parameter specifies the size of this work space.

This area does not stay permanently allocated, but is allocated and released for every SQL or XQuery statement handled. Note that for dynamic SQL or XQuery statements, this work area will be used during execution of your program; whereas, for static SQL or XQuery statements, it is used during the bind process but not during program execution.

Recommendation: In most cases the default value of this parameter will be acceptable. If you have very large SQL or XQuery statements and the database manager issues an error (that the statement is too complex) when it attempts to optimize a statement, you should increase the value of this parameter in regular increments (such as 256 or 1024) until the error situation is resolved.

Related reference:

- "sortheap - Sort heap size " on page 400
- "applheapsz - Application heap size " on page 397
- "stat_heap_sz - Statistics heap size " on page 402
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Agent/application communication memory

The following parameters affect the amount of memory that is allocated to allow data to be passed between your application and agent processes:

- "aslheapsz - Application support layer heap size "
- "min_dec_div_3 - Decimal division scale to 3 " on page 405
- "rqrioblk - Client I/O block size " on page 406

aslheapsz - Application support layer heap size

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 15 [1 – 524 288]

Unit of Measure	Pages (4 KB)
When Allocated	When the database manager agent process is started for the local application
When Freed	When the database manager agent process is terminated

The application support layer heap represents a communication buffer between the local application and its associated agent. This buffer is allocated as shared memory by each database manager agent that is started.

If the request to the database manager, or its associated reply, do not fit into the buffer they will be split into two or more send-and-receive pairs. The size of this buffer should be set to handle the majority of requests using a single send-and-receive pair. The size of the request is based on the storage required to hold:

- The input SQLDA
- All of the associated data in the SQLVARs
- The output SQLDA
- Other fields which do not generally exceed 250 bytes.

In addition to this communication buffer, this parameter is also used for two other purposes:

- It is used to determine the I/O block size when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.
- It is used to determine the communication size between agents and db2fmp processes. (A db2fmp process can be a user-defined function or a fenced stored procedure.) The number of bytes is allocated from shared memory for each db2fmp process or thread that is active on the system.

The data sent from the local application is received by the database manager into a set of contiguous memory allocated from the query heap. The *aslheapsz* parameter is used to determine the initial size of the query heap (for both local and remote clients). The maximum size of the query heap is defined by the *query_heap_sz* parameter.

Recommendation: If your application's requests are generally small and the application is running on a memory constrained system, you might wish to reduce the value of this parameter. If your queries are generally very large, requiring more than one send and receive request, and your system is not constrained by memory, you might wish to increase the value of this parameter.

Use the following formula to calculate a minimum number of pages for *aslheapsz*:

```
aslheapsz >= ( sizeof(input SQLDA)
             + sizeof(each input SQLVAR)
             + sizeof(output SQLDA)
             + 250 ) / 4096
```

where sizeof(x) is the size of x in bytes that calculates the number of pages of a given input or output value.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks might yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4 096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks might also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

Related reference:

- "query_heap_sz - Query heap size " on page 398
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

min_dec_div_3 - Decimal division scale to 3

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Yes, No]

The *min_dec_div_3* database configuration parameter is provided as a quick way to enable a change to computation of the scale for decimal division in SQL. *min_dec_div_3* can be set to ″Yes″ or ″No″. The default value for *min_dec_div_3* is ″No″.

The *min_dec_div_3* database configuration parameter changes the resulting scale of a decimal arithmetic operation involving division. If the value is ″No″, the scale is calculated as 31-p+s-s′. If set to ″Yes″, the scale is calculated as MAX(3, 31-p+s-s′). This causes the result of decimal division to always have a scale of at least 3. Precision is always 31.

Changing this database configuration parameter might cause changes to applications for existing databases. This can occur when the resulting scale for decimal division would be impacted by changing this database configuration parameter. Listed below are some possible scenarios that might impact applications. These scenarios should be considered before changing the *min_dec_div_3* on a database server with existing databases.

- If the resulting scale of one of the view columns is changed, a view that is defined in an environment with one setting could fail with SQLCODE -344 when referenced after the database configuration parameter is changed. The message SQL0344N refers to recursive common table expressions, however, if the object name (first token) is a view, then you will need to drop the view and create it again to avoid this error.
- A static package will not change behavior until the package is rebound, either implicitly or explicitly. For example, after changing the value from NO to YES, the additional scale digits might not be included in the results until rebind occurs. For any changed static packages, an explicit REBIND command can be used to force a rebind.

- A check constraint involving decimal division might restrict some values that were previously accepted. Such rows now violate the constraint but will not be detected until one of the columns involved in the check constraint row is updated or the SET INTEGRITY statement with the IMMEDIATE CHECKED option is processed. To force checking of such a constraint, perform an ALTER TABLE statement in order to drop the check constraint and then perform an ALTER TABLE statement to add the constraint again.

Note: *min_dec_div_3* also has the following limitations:

1. The command `GET DB CFG FOR DBNAME` will not display the *min_dec_div_3* setting. The best way to determine the current setting is to observe the side-effect of a decimal division result. For example, consider the following statement:

   ```
   VALUES (DEC(1,31,0)/DEC(1,31,5))
   ```

   If this statement returns sqlcode SQL0419N, the database does not have *min_dec_div_3* support, or it is set to "No". If the statement returns `1.000`, *min_dec_div_3* is set to "Yes".
2. *min_dec_div_3* does not appear in the list of configuration keywords when you run the following command: `? UPDATE DB CFG`

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

rqrioblk - Client I/O block size

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 32 767 [4 096 – 65 535]

Unit of Measure Bytes

When Allocated

- When a remote client application issues a connection request for a server database
- When a blocking cursor is opened, additional blocks are opened at the client

When Freed

- When the remote application disconnects from the server database
- When the blocking cursor is closed

This parameter specifies the size of the communication buffer between remote applications and their database agents on the database server. When a database

client requests a connection to a remote database, this communication buffer is allocated on the client. On the database server, a communication buffer of 32 767 bytes is initially allocated, until a connection is established and the server can determine the value of *rqrioblk* at the client. Once the server knows this value, it will reallocate its communication buffer if the client's buffer is not 32 767 bytes.

In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

Recommendation: For non-blocking cursors, a reason for increasing the value of this parameter would be if the data (for example, large object data) to be transmitted by a single query statement is so large that the default value is insufficient.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks might yield better performance if the number or size of rows being transferred is large (for example, if the amount of data is greater than 4 096 bytes). However, there is a trade-off in that larger record blocks increase the size of the working set memory for each connection.

Larger record blocks might also cause more fetch requests than are actually required by the application. You can control the number of fetch requests using the OPTIMIZE FOR clause on the SELECT statement in your application.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Database manager instance memory

The following parameters affect memory that is allocated and used at an instance level:

- "audit_buf_sz - Audit buffer size "
- "dir_cache - Directory cache support " on page 408
- "instance_memory - Instance memory " on page 410
- "java_heap_sz - Maximum Java interpreter heap size " on page 410
- "mon_heap_sz - Database system monitor heap size " on page 411

audit_buf_sz - Audit buffer size

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 0 [0 – 65 000]

Unit of Measure Pages (4 KB)

When Allocated When DB2 is started

When Freed When DB2 is stopped

This parameter specifies the size of the buffer used when auditing the database.

The default value for this parameter is zero (0). If the value is zero (0), the audit buffer is not used. If the value is greater than zero (0), space is allocated for the audit buffer where the audit records will be placed when they are generated by the audit facility. The value times 4 KB pages is the amount of space allocated for the audit buffer. The audit buffer cannot be allocated dynamically; DB2 must be stopped and then restarted before the new value for this parameter takes effect.

By changing this parameter from the default to some value larger than zero (0), the audit facility writes records to disk asynchronously compared to the execution of the statements generating the audit records. This improves DB2 performance over leaving the parameter value at zero (0). The value of zero (0) means the audit facility writes records to disk synchronously with (at the same time as) the execution of the statements generating the audit records. The synchronous operation during auditing decreases the performance of applications running in DB2.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

dir_cache - Directory cache support

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Yes [Yes; No]

When Allocated

- When an application issues its first connect, the application directory cache is allocated

- When a database manager instance is started (db2start), the server directory cache is allocated.

When Freed

- When an the application process terminates, the application directory cache is freed
- When a database manager instance is stopped (db2stop), the server directory cache is freed.

By setting *dir_cache* to Yes the database, node and DCS directory files will be cached in memory. The use of the directory cache reduces connect costs by eliminating directory file I/O and minimizing the directory searches required to retrieve directory information. There are two types of directory caches:

- An application directory cache that is allocated and used for each application process on the machine at which the application is running.
- A server directory cache that is allocated and used for some of the internal database manager processes.

For application directory caches, when an application issues its first connect, each directory file is read and the information is cached in private memory for this application. The cache is used by the application process on subsequent connect requests and is maintained for the life of the application process. If a database is not found in the application directory cache, the directory files are searched for the information, but the cache is not updated. If the application modifies a directory entry, the next connect within that application will cause the cache for this application to be refreshed. The application directory cache for other applications will not be refreshed. When the application process terminates, the cache is freed. (To refresh the directory cache used by a command line processor session, issue a `db2 terminate` command.)

For server directory caches, when a database manager instance is started (`db2start`), each directory file is read and the information is cached in the server memory. This cache is maintained until the instance is stopped (`db2stop`). If a directory entry is not found in this cache, the directory files are searched for the information. This server directory cache is never refreshed during the time the instance is running.

Recommendation: Use directory caching if your directory files do not change frequently and performance is critical.

In addition, on remote clients, directory caching can be beneficial if your applications issue several different connection requests. In this case, caching reduces the number of times a single application must read the directory files.

Directory caching can also improve the performance of taking database system monitor snapshots. In addition, you should explicitly reference the database name on the snapshot call, instead of using database aliases.

Note: Errors might occur when performing snapshot calls if directory caching is turned on and if databases are cataloged, uncataloged, created, or dropped after the database manager is started.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

instance_memory - Instance memory

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Automatic [8 — 4 294 967 295]

Unit of Measure Pages (4 KB)

When Allocated When the instance is started

When Freed When the instance is stopped

This parameter specifies the amount of memory that should be reserved for instance management. This includes memory areas that describe the databases on the instance.

If you set this parameter to AUTOMATIC, DB2 will calculate the amount of instance memory needed for the current configuration. DB2 will also allocate some additional unreserved memory that any heap in the instance shared memory region can use to satisfy peak memory requirements. Other operations, such as dynamic configuration updates, also have access to this unreserved memory. The **db2pd** command, with the **-memsets** option, can be used to monitor the amount of unused memory left in the instance shared memory region.

Related reference:

- "maxagents - Maximum number of agents " on page 426
- "numdb - Maximum number of concurrently active databases including host and iSeries databases " on page 502
- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

java_heap_sz - Maximum Java interpreter heap size

Configuration Type Database manager

Applies to

- Database server with local and remote clients

- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

32-bit platforms
512 [0 - 524 288]

64-bit platforms
1024[0 - 524 288]

Unit of Measure Pages (4 KB)

When Allocated When a Java™ stored procedure or UDF starts

When Freed When the db2fmp process (fenced) or the db2agent process (trusted) terminates.

This parameter determines the maximum size of the heap that is used by the Java interpreter started to service Java DB2 stored procedures and UDFs.

There is one heap for each DB2 process (one for each agent or subagent on UNIX-based platforms, and one for each instance on other platforms). There is one heap for each fenced UDF and fenced stored procedure process. There is one heap per agent (not including sub-agents) for trusted routines. There is one heap per db2fmp process running a Java stored procedure. For multithreaded db2fmp processes, multiple applications using threadsafe fenced routines are serviced from a single heap. In all situations, only the agents or processes that run Java UDFs or stored procedures ever allocate this memory. On partitioned database systems, the same value is used at each database partition.

Related reference:

- "jdk_path - Software Developer's Kit for Java installation path " on page 501
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

mon_heap_sz - Database system monitor heap size

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX 90 [0 – 60 000]

	Windows Database server with local and remote clients 66 [0 – 60 000] **Windows Database server with local clients** 46 [0 – 60 000]
Unit of Measure	Pages (4 KB)
When Allocated	When the database manager is started with the *db2start* command
When Freed	When the database manager is stopped with the *db2stop* command

This parameter determines the amount of the memory, in pages, to allocate for database system monitor data. Memory is allocated from the monitor heap when you perform database monitoring activities such as taking a snapshot, turning on a monitor switch, resetting a monitor, or activating an event monitor.

A value of zero prevents the database manager from collecting database system monitor data.

Recommendation: The amount of memory required for monitoring activity depends on the number of monitoring applications (applications taking snapshots or event monitors), which switches are set, and the level of database activity.

If the configured memory in this heap runs out and no more unreserved memory is available in the instance shared memory region, one of the following will occur:

- When the first application connects to the database for which this event monitor is defined, an error message is written to the administration notification log.
- If an event monitor being started dynamically using the SET EVENT MONITOR statement fails, an error code is returned to your application.
- If a monitor command or API subroutine fails, an error code is returned to your application.

Related concepts:

- "Database system monitor memory requirements" in *System Monitor Guide and Reference*

Related reference:

- "dft_monswitches - Default database system monitor switches " on page 497
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Locks

The following parameters influence how locking is managed in your environment:

- "dlchktime - Time interval for checking deadlock " on page 413
- "locktimeout - Lock timeout " on page 413
- "maxlocks - Maximum percent of lock list before escalation " on page 414

See also "locklist - Maximum storage for lock list " on page 383.

dlchktime - Time interval for checking deadlock

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	10 000 (10 seconds) [1 000 – 600 000]
Unit of Measure	Milliseconds

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

The deadlock check interval defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

Notes:

1. In a partitioned database environment, this parameter applies to the catalog node only.
2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

Recommendation: Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease run-time performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that *maxlocks* and *locklist* are set appropriately to avoid unnecessary lock escalation, which can result in more lock contention and as a result, more deadlock situations.

Related reference:

- "locklist - Maximum storage for lock list " on page 383
- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

locktimeout - Lock timeout

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	-1 [-1; 0 – 32 767]
Unit of Measure	Seconds

This parameter specifies the number of seconds that an application will wait to obtain a lock. This helps avoid global deadlocks for applications.

If you set this parameter to `0`, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

If you set this parameter to `-1`, lock timeout detection is turned off. In this situation a lock will be waited for (if one is not available at the time of the request) until either of the following:

- The lock is granted
- A deadlock occurs.

Recommendation: In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

When working with Data Links Manager, if you see lock timeouts in the administration notification log of the Data Links Manager (dlfm) instance, then you should increase the value of *locktimeout*. You should also consider increasing the value of *locklist*.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a user leaving their workstation). You should set it high enough so valid lock requests do not time-out because of peak workloads, during which time, there is more waiting for locks.

You can use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected.

High values of the *lock_timeout* (number of lock timeouts) monitor element can be caused by:

- Too low a value for this configuration parameter.
- An application (transaction) that is holding locks for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (from row-level to a table-level lock).

Related concepts:

- "Lock waits and timeouts" on page 68

Related reference:

- "lock_timeouts - Number of Lock Timeouts monitor element" in *System Monitor Guide and Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "locklist - Maximum storage for lock list " on page 383
- "maxlocks - Maximum percent of lock list before escalation " on page 414

maxlocks - Maximum percent of lock list before escalation

Configuration Type Database

Parameter Type Configurable online

Propagation Class	Immediate	
Default [Range]		
	UNIX	Automatic [1 – 100]
	Windows	Automatic [1 – 100]
Unit of Measure	Percentage	

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the *maxlocks* value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of *maxlocks*. The *maxlocks* parameter multiplied by the *maxappls* parameter cannot be less than 100.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

The value of *locklist* is tuned together with the *maxlocks* parameter, therefore disabling self tuning of the *locklist* parameter automatically disables self tuning of the *maxlocks* parameter. Enabling self tuning of the *locklist* parameter automatically enables self tuning of the *maxlocks* parameter.

Recommendation: The following formula allows you to set *maxlocks* to allow an application to hold twice the average number of locks:

```
maxlocks = 2 * 100 / maxappls
```

Where `2` is used to achieve twice the average and `100` represents the largest percentage value allowed. If you have only a few applications that run concurrently, you could use the following formula as an alternative to the first formula:

```
maxlocks = 2 * 100 / (average number of applications running
concurrently)
```

One of the considerations when setting *maxlocks* is to use it in conjunction with the size of the lock list (*locklist*). The actual limit of the number of locks held by an application before lock escalation occurs is:

`maxlocks * locklist * 4 096 / (100 * 48)` on a 32-bit system

`maxlocks * locklist * 4 096 / (100 * 80)` on a 64-bit system HP-UX/PA-RISC environment

`maxlocks * locklist * 4 096 / (100 * 64)` on other 64-bit systems

Where 4 096 is the number of bytes in a page, 100 is the largest percentage value allowed for *maxlocks*, and 48 is the number of bytes per lock on a 32-bit system, 80 is the number of bytes per lock on a HP-UX/PA-RISC 64-bit system, and 64 is the number of bytes per lock on other 64-bit systems. If you know that one of your applications requires 1 000 locks, and you do not want lock escalation to occur, then you should choose values for *maxlocks* and *locklist* in this formula so that the result is greater than 1 000. (Using 10 for *maxlocks* and 100 for *locklist*, this formula results in greater than the 1 000 locks needed.)

If *maxlocks* is set too low, lock escalation happens when there is still enough lock space for other concurrent applications. If *maxlocks* is set too high, a few applications can consume most of the lock space, and other applications will have to perform lock escalation. The need for lock escalation in this case results in poor concurrency.

You can use the database system monitor to help you track and tune this configuration parameter.

Related concepts:

- "Self tuning memory" on page 255

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "locklist - Maximum storage for lock list " on page 383
- "maxappls - Maximum number of active applications " on page 427
- "self_tuning_mem- Self tuning memory " on page 389

I/O and storage

The following parameters can influence I/O and storage costs related to the operation of your database:

- "chngpgs_thresh - Changed pages threshold "
- "dft_extent_sz - Default extent size of table spaces " on page 417
- "dft_prefetch_sz - Default prefetch size " on page 418
- "num_iocleaners - Number of asynchronous page cleaners " on page 419
- "num_ioservers - Number of I/O servers " on page 420
- "numsegs - Default number of SMS containers " on page 421
- "seqdetect - Sequential detection flag " on page 421

chngpgs_thresh - Changed pages threshold

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	60 [5 – 99]
Unit of Measure	Percentage

Asynchronous page cleaners will write changed pages from the buffer pool (or the buffer pools) to disk before the space in the buffer pool is required by a database agent. As a result, database agents should not have to wait for changed pages to

be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

You can use this parameter to specify the level (percentage) of changed pages at which the asynchronous page cleaners will be started, if they are not currently active. When the page cleaners are started, they will build a list of the pages to write to disk. Once they have completed writing those pages to disk, they will become inactive again and wait for the next trigger to start.

In a read-only (for example, query) environment, these page cleaners are not used.

When the DB2_USE_ALTERNATE_PAGE_CLEANING registry variable is set (that is, the alternate method of page cleaning is used), the *chngpgs_thresh* parameter has no effect, and DB2 automatically determines how many dirty pages to maintain in the buffer pool.

Recommendation: For databases with a heavy update transaction workload, you can generally ensure that there are enough clean pages in the buffer pool by setting the parameter value to be equal-to or less-than the default value. A percentage larger than the default can help performance if your database has a small number of very large tables.

Related reference:

- "num_iocleaners - Number of asynchronous page cleaners " on page 419
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

dft_extent_sz - Default extent size of table spaces

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	32 [2 – 256]
Unit of Measure	Pages

When a table space is created, `EXTENTSIZE n` can be optionally specified, where `n` is the extent size. If you do not specify the extent size on the CREATE TABLESPACE statement, the database manager uses the value given by this parameter.

Recommendation: In many cases, you will want to explicitly specify the extent size when you create the table space. Before choosing a value for this parameter, you should understand how you would explicitly choose an extent size for the CREATE TABLESPACE statement.

Related concepts:

- "Extent size" in *Administration Guide: Planning*

Related reference:

- "CREATE TABLESPACE statement" in *SQL Reference, Volume 2*
- "dft_prefetch_sz - Default prefetch size " on page 418
- "GET DATABASE CONFIGURATION command" in *Command Reference*

- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

dft_prefetch_sz - Default prefetch size

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Automatic [0 — 32 767]
Unit of Measure	Pages

When a table space is created, PREFETCHSIZE *n* can optionally be specified, where *n* represents the number of pages the database manager will read if prefetching is being performed. If you do not specify the prefetch size on invocation of the CREATE TABLESPACE statement, the database manager uses the current value of the *dft_prefetch_sz* parameter.

If a table space is created with AUTOMATIC DFT_PREFETCH_SZ, the prefetch size of the table space will become AUTOMATIC, which means that DB2 will automatically calculate and update the prefetch size of the table space, using the following equation:

```
prefetch size = (# containers) * (# physical spindles) * extent size
```

where the number of physical spindles defaults to 1 and can be specified through the DB2 registry variable DB2_PARALLEL_IO. This calculation is performed:

- At database start-up time
- When a table space is first created with AUTOMATIC prefetch size
- When the number of containers for a table space changes through execution of an ALTER TABLESPACE statement
- When the prefetch size for a table space is updated to be AUTOMATIC through execution of an ALTER TABLESPACE statement

The AUTOMATIC state of the prefetch size can be turned on or off as soon as the prefetch size is updated manually through invocation of the ALTER TABLESPACE statement.

Recommendation: Using system monitoring tools, you can determine if your CPU is idle while the system is waiting for I/O. Increasing the value of this parameter can help if the table spaces being used do not have a prefetch size defined for them.

This parameter provides the default for the entire database, and it might not be suitable for all table spaces within the database. For example, a value of 32 might be suitable for a table space with an extent size of 32 pages, but not suitable for a table space with an extent size of 25 pages. Ideally, you should explicitly set the prefetch size for each table space.

To help minimize I/O for table spaces defined with the default extent size (*dft_extent_sz*), you should set this parameter as a factor or whole multiple of the value of the *dft_extent_sz* parameter. For example, if the *dft_extent_sz* parameter is 32, you could set *dft_prefetch_sz* to 16 (a fraction of 32) or to 64 (a whole multiple of 32). If the prefetch size is a multiple of the extent size, the database manager might perform I/O in parallel, if the following conditions are true:

- The extents being prefetched are on different physical devices

- Multiple I/O servers are configured (*num_ioservers*).

Related reference:
- "ALTER TABLESPACE statement" in *SQL Reference, Volume 2*
- "CREATE TABLESPACE statement" in *SQL Reference, Volume 2*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

num_iocleaners - Number of asynchronous page cleaners

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Automatic[0 – 255]
Unit of Measure	Counter

This parameter allows you to specify the number of asynchronous page cleaners for a database. These page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. As a result, database agents should not have to wait for changed pages to be written out so that they might use the space in the buffer pool. This improves overall performance of the database applications.

If you set the parameter to zero (0), no page cleaners are started and as a result, the database agents will perform all of the page writes from the buffer pool to disk. This parameter can have a significant performance impact on a database stored across many physical storage devices, since in this case there is a greater chance that one of the devices will be idle. If no page cleaners are configured, your applications might encounter periodic log full conditions.

If this parameter is set to AUTOMATIC, the number of page cleaners started will be based on the number of CPUs configured on the current machine, as well as the number of local logical database partitions in a partitioned database environment. There will always be at least one page cleaner started when this parameter is set to AUTOMATIC.

The number of page cleaners to start when this parameter is set to AUTOMATIC will be calculated using the following formula:

```
number of page cleaners = max( ceil( # CPUs / # local logical DPs ) – 1, 1 )
```

This formula ensures that the number of page cleaners is distributed almost evenly across your logical database partitions, and that there are no more page cleaners than there are CPUs.

If the applications for a database primarily consist of transactions that update data, an increase in the number of cleaners will speed up performance. Increasing the page cleaners will also decrease recovery time from soft failures, such as power outages, because the contents of the database on disk will be more up-to-date at any given time.

Recommendation: Consider the following factors when setting the value for this parameter:
- Application type

  - If it is a query-only database that will not have updates, set this parameter to be zero (0). The exception would be if the query work load results in many TEMP tables being created (you can determine this by using the explain utility).
  - If transactions are run against the database, set this parameter to be between one and the number of physical storage devices used for the database.
- Workload

  Environments with high update transaction rates might require more page cleaners to be configured.
- Buffer pool sizes

  Environments with large buffer pools might also require more page cleaners to be configured.

You can use the database system monitor to help you tune this configuration parameter using information from the event monitor about write activity from a buffer pool:

- The parameter can be reduced if both of the following conditions are true:
  - *pool_data_writes* is approximately equal to *pool_async_data_writes*
  - *pool_index_writes* is approximately equal to *pool_async_index_writes*.
- The parameter should be increased if either of the following conditions are true:
  - *pool_data_writes* is much greater than *pool_async_data_writes*
  - *pool_index_writes* is much greater than *pool_async_index_writes*.

Related reference:

- "pool_async_data_writes - Buffer Pool Asynchronous Data Writes monitor element" in *System Monitor Guide and Reference*
- "pool_async_index_writes - Buffer Pool Asynchronous Index Writes monitor element" in *System Monitor Guide and Reference*
- "pool_data_writes - Buffer Pool Data Writes monitor element" in *System Monitor Guide and Reference*
- "pool_index_writes - Buffer Pool Index Writes monitor element" in *System Monitor Guide and Reference*
- "chngpgs_thresh - Changed pages threshold " on page 416
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

num_ioservers - Number of I/O servers

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Automatic [1 – 255]
Unit of Measure	Counter
When Allocated	When an application connects to a database
When Freed	When an application disconnects from a database

I/O servers, also called prefetchers, are used on behalf of the database agents to perform prefetch I/O and asynchronous I/O by utilities such as backup and restore. This parameter specifies the number of I/O servers for a database. No

more than this number of I/Os for prefetching and utilities can be in progress for a database at any time. An I/O server waits while an I/O operation that it initiated is in progress. Non-prefetch I/Os are scheduled directly from the database agents and as a result are not constrained by *num_ioservers*.

If this parameter is set to AUTOMATIC, the number of prefetchers started will be based on the parallelism settings of the table spaces in the current database partition. (Parallelism settings are controlled by the DB2_PARALLEL_IO environment variable.) For each DMS table space, the value of this parallelism setting will be multiplied by the maximum number of containers in the table space stripe set. For each SMS table space, the value of this parallelism setting will be multiplied by the number of containers in the table space. The largest result over all table spaces in the current database partition will be used as the number of prefetchers to start. There will always be at least three prefetchers started when this parameter is set to AUTOMATIC.

When this parameter is set to AUTOMATIC, the number of prefetchers to start will be calculated at database activation time based on the following formula:

```
number of prefetchers = max( max over all table spaces
( parallelism setting * [SMS: # containers; DMS: max # containers in stripe set] ), 3 )
```

Recommendation: In order to fully exploit all the I/O devices in the system, a good value to use is generally one or two more than the number of physical devices on which the database resides. It is better to configure additional I/O servers, since there is minimal overhead associated with each I/O server and any unused I/O servers will remain idle.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "dft_prefetch_sz - Default prefetch size " on page 418
- "seqdetect - Sequential detection flag " on page 421

numsegs - Default number of SMS containers

Configuration Type	Database
Parameter Type	Informational
Unit of Measure	Counter

This parameter, which only applies to SMS table spaces, indicates the number of containers that will be created within the default table spaces. This parameter will show the information used when you created your database, whether it was specified explicitly or implicitly on the CREATE DATABASE command. The CREATE TABLESPACE statement **does not** use this parameter in any way.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

seqdetect - Sequential detection flag

Configuration Type	Database
Parameter Type	Configurable online

Propagation Class Immediate

Default [Range] Yes [Yes; No]

The database manager can monitor I/O and if sequential page reading is occurring the database manager can activate I/O prefetching. This type of sequential prefetch is known as *sequential detection*. You can use the *seqdetect* configuration parameter to control whether the database manager should perform sequential detection.

If this parameter is set to No, prefetching takes place only if the database manager knows it will be useful, for example table sorts, table scans, or list prefetch.

Recommendation: In most cases, you should use the default value for this parameter. Try turning sequential detection off, only if other tuning efforts were unable to correct serious query performance problems.

Related reference:

- "dft_prefetch_sz - Default prefetch size " on page 418
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Agents

The following parameters can influence the number of applications that can be run concurrently and achieve optimal performance:

- "agentpri - Priority of agents "
- "avg_appls - Average number of active applications " on page 424
- "max_connections - Maximum number of client connections " on page 424
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "maxagents - Maximum number of agents " on page 426
- "maxappls - Maximum number of active applications " on page 427
- "maxcagents - Maximum number of concurrent agents " on page 428
- "maxfilop - Maximum database files open per application " on page 429
- "maxtotfilop - Maximum total files open " on page 430
- "num_initagents - Initial number of agents in pool " on page 431
- "num_poolagents - Agent pool size " on page 431

agentpri - Priority of agents

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

AIX -1 (system) [41 - 125]

Other UNIX
-1 (system) [41 - 128]

Windows

-1 (system) [0 - 6]

This parameter controls the priority given both to all agents, and to other database manager instance processes and threads, by the operating system scheduler. In a partitioned database environment, this also includes both coordinating and subagents, the parallel system controllers, and the FCM daemons. This priority determines how CPU time is given to the DB2 processes, agents, and threads relative to the other processes and threads running on the machine. When the parameter is set to -1 or system, no special action is taken and the database manager is scheduled in the normal way that the operating system schedules all processes and threads. When the parameter is set to a value other than -1 or system, the database manager will create its processes and threads with a static priority set to the value of the parameter. Therefore, this parameter allows you to control the priority with which the database manager processes and threads will execute on your machine.

You can use this parameter to increase database manager throughput. The values for setting this parameter are dependent on the operating system on which the database manager is running. For example, in a UNIX-based environment, numerically low values yield high priorities. When the parameter is set to a value between 41 and 125, the database manager creates its agents with a UNIX static priority set to the value of the parameter. This is important in UNIX-based environments because numerically low values yield high priorities for the database manager, but other processes (including applications and users) might experience delays because they cannot obtain enough CPU time. You should balance the setting of this parameter with the other activity expected on the machine.

Recommendation: The default value should be used initially. This value provides a good compromise between response time to other users/applications and database manager throughput.

Note: On the Solaris Operating Environment, you should not change the default value (−1). Changing the default value sets the priority of DB2 processes to real-time, which can monopolize all available resources on the system.

If database performance is a concern, you can use benchmarking techniques to determine the optimum setting for this parameter. You should take care when increasing the priority of the database manager because performance of other user processes can be severely degraded, especially when the CPU utilization is very high. Increasing the priority of the database manager processes and threads can have significant performance benefits.

Note: If you set this parameter to a non-default value on UNIX-based platforms, you cannot use the governor to alter agent priorities.

Related reference:

- “GET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE MANAGER CONFIGURATION command” in *Command Reference*

avg_appls - Average number of active applications

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	Automatic [1 – maxappls]
Unit of Measure	Counter

This parameter is used by the query optimizer to help estimate how much buffer pool will be available at run-time for the access plan chosen.

By default, this parameter is set to Automatic for databases created in IBM DB2 Version 9.1. When *avg_appls* is set to AUTOMATIC, the optimizer uses a value of 3 if the registry variable DB2_WORKLOAD=SAP and if the value of *max_appls* is greater than 3. Otherwise, when *avg_appls* is set to AUTOMATIC, the optimizer uses a value of 1.

Recommendation: When running DB2 in a multi-user environment, particularly with complex queries and a large buffer pool, you might want the query optimizer to know that multiple query users are using your system so that the optimizer should be more conservative in assumptions of buffer pool availability.

When setting this parameter, you should estimate the number of complex query applications that typically use the database. This estimate should exclude all light OLTP applications. If you have trouble estimating this number, you can multiply the following:

- An average number of all applications running against your database. The database system monitor can provide information about the number of applications at any given time and using a sampling technique, you can calculate an average over a period of time. The information from the database system monitor includes both OLTP and non-OLTP applications.
- Your estimate of the percentage of complex query applications.

As with adjusting other configuration parameters that affect the optimizer, you should adjust this parameter in small increments. This allows you to minimize path selection differences.

You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Related reference:

- "maxappls - Maximum number of active applications " on page 427
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

max_connections - Maximum number of client connections

Configuration Type	Database manager
Parameter Type	Configurable
Default [Range]	-1 (*max_coordagents*) [-1; 1 — 64 000]

When the Concentrator is off, this parameter indicates the maximum number of client connections allowed per database partition. The Concentrator is off when *max_connections* is equal to or less than *max_coordagents*. The Concentrator is on when *max_connections* is greater than *max_coordagents*.

This parameter controls the maximum number of applications that can be connected to the instance. Typically, each application is assigned a coordinator agent. An agent facilitates the operations between the application and the database. When the default value for this parameter is used, the concentrator feature is not activated. As a result, each agent operates with its own private memory and shares database manager and database global resources such as the buffer pool with other agents. When the parameter is set to a value greater than the default, the concentrator feature is activated. The intent of the concentrator is to reduce the server resources per client application to a point where a DB2 Connect gateway can handle greater than 10 000 client connections.

A value of -1 indicates that the limit is *max_coordagents*.

In previous versions of DB2, this parameter was called *max_logicagents*.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

max_coordagents - Maximum number of coordinating agents

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (*maxagents* – *num_initagents*)

[-1, 0 – *maxagents*]

For partitioned database environments and environments in which *intra_parallel* is set to Yes, the default is *maxagents* minus *num_initagents*; otherwise, the default is *maxagents*. This ensures that, in non-partitioned database environments, *max_coordagents* always equals *maxagents*, unless the system is configured for intra-partition parallelism.

If you do not have a partitioned database environment, and have not enabled the *intra_parallel* parameter, *max_coordagents* must equal *maxagents*.

When the Concentrator is off, that is, when *max_connections* is equal to *max_coordagents*, this parameter determines the maximum number of coordinating agents that can exist at one time on a server in a partitioned or non-partitioned database environment.

One coordinating agent is acquired for each local or remote application that connects to a database or attaches to an instance. Requests that require an instance attachment include CREATE DATABASE, DROP DATABASE, and Database System Monitor commands.

When the Concentrator is on, that is, when *max_connections* is greater than *max_coordagents*, there might be more connections than coordinator agents to service them. An application is in an active state only if there is a coordinator agent servicing it. Otherwise, the application is in an inactive state. Requests from an active application will be serviced by the database coordinator agent (and subagents in SMP or MPP configurations). Requests from an inactive application will be queued until a database coordinator agent is assigned to service the application, when the application becomes active. As a result, this parameter can be used to control the load on the system.

Related reference:

- "num_initagents - Initial number of agents in pool " on page 431
- "num_poolagents - Agent pool size " on page 431
- "maxagents - Maximum number of agents " on page 426
- "intra_parallel - Enable intra-partition parallelism " on page 491
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

maxagents - Maximum number of agents

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	200 [1 – 64 000] 400 [1 – 64 000] on Partitioned database server with local and remote clients
Unit of Measure	Counter

This parameter indicates the maximum number of database manager agents, whether coordinator agents or subagents, available at any given time to accept application requests. If you want to limit the number of coordinating agents, use the *max_coordagents* parameter.

This parameter can be useful in memory constrained environments to limit the total memory usage of the database manager, because each additional agent requires additional memory.

Recommendation: The value of *maxagents* should be at least the sum of the values for *maxappls* in each database allowed to be accessed concurrently. If the number of databases is greater than the *numdb* parameter, then the safest course is to use the product of *numdb* with the largest value for *maxappls*.

Each additional agent requires some resource overhead that is allocated at the time the database manager is started.

Related reference:

- "maxappls - Maximum number of active applications " on page 427
- "maxcagents - Maximum number of concurrent agents " on page 428
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "fenced_pool - Maximum number of fenced processes " on page 432
- "num_poolagents - Agent pool size " on page 431
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

maxappls - Maximum number of active applications

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Automatic [Automatic; 1 – 60 000]
Unit of Measure	Counter

This parameter specifies the maximum number of concurrent applications that can be connected (both local and remote) to a database. Since each application that attaches to a database causes some private memory to be allocated, allowing a larger number of concurrent applications will potentially use more memory.

Setting *maxappls* to *automatic* has the effect of allowing any number of connected applications. DB2 will dynamically allocate the resources it needs to support new applications.

If you do not want to set this parameter to *automatic*, the value of this parameter must be equal to or greater than the sum of the connected applications, plus the number of these same applications that might be concurrently in the process of completing a two-phase commit or rollback. Then add to this sum the anticipated number of indoubt transactions that might exist at any one time.

When an application attempts to connect to a database, but *maxappls* has already been reached, an error is returned to the application indicating that the maximum number of applications have been connected to the database.

As more applications use the Data Links Manager, the value of *maxappls* should be increased. Use the following formula to compute the value you need:

```
<maxappls> = 5 * (number of nodes) + (peak number of active applications
                  using Data Links Manager)
```

The maximum supported value for Data Links Manager is 2 000.

In a partitioned database environment, this is the maximum number of applications that can be concurrently active against a database partition. This parameter limits the number of active applications against the database partition on a database partition server, regardless of whether the server is the coordinator node for the application or not. The catalog node in a partitioned database environment requires a higher value for *maxappls* than is the case for other types of environments because, in the partitioned database environment, every application requires a connection to the catalog node.

Recommendation: Increasing the value of this parameter without lowering the *maxlocks* parameter or increasing the *locklist* parameter could cause you to reach the database limit on locks (*locklist*) rather than the application limit and as a result cause pervasive lock escalation problems.

To a certain extent, the maximum number of applications is also governed by *maxagents*. An application can only connect to the database, if there is an available connection (*maxappls*) as well as an available agent (*maxagents*). In addition, the maximum number of applications is also controlled by the *max_coordagents* configuration parameter, because no new applications (that is, coordinator agents) can be started if *max_coordagents* has been reached.

Related tasks:

- "Resolving indoubt transactions manually" in *Administration Guide: Planning*

Related reference:

- "maxagents - Maximum number of agents " on page 426
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "maxlocks - Maximum percent of lock list before escalation " on page 414
- "locklist - Maximum storage for lock list " on page 383
- "avg_appls - Average number of active applications " on page 424
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

maxcagents - Maximum number of concurrent agents

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (*max_coordagents*) [-1; 1 – *max_coordagents*]

Unit of Measure Counter

The maximum number of database manager agents that can be concurrently executing a database manager transaction. This parameter is used to control the load on the system during periods of high simultaneous application activity. For example, you might have a system requiring a large number of connections but with a limited amount of memory to serve those connections. Adjusting this parameter can be useful in such an environment, where a period of high simultaneous activity could cause excessive operating system paging.

This parameter does not limit the number of applications that can have connections to a database. It only limits the number of database manager agents that can be processed concurrently by the database manager at any one time, thereby limiting the usage of system resources during times of peak processing.

A value of –1 indicates that the limit is *max_coordagents*.

Recommendation: In most cases the default value for this parameter will be acceptable. In cases where the high concurrency of applications is causing problems, you can use benchmark testing to tune this parameter to optimize the performance of the database.

Related reference:

- "maxappls - Maximum number of active applications " on page 427
- "maxagents - Maximum number of agents " on page 426
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

maxfilop - Maximum database files open per application

Configuration Type	Database	
Parameter Type	Configurable Online	
Propagation Class	Transaction boundary	
Default [Range]		
	UNIX	64 [2 – 1950]
	Windows	64 [2 – 32 768]
Unit of Measure	Counter	

This parameter specifies the maximum number of file handles that can be open for each database agent. If opening a file causes this value to be exceeded, some files in use by this agent are closed. If *maxfilop* is too small, the overhead of opening and closing files so as not to exceed this limit will become excessive and might degrade performance.

Both SMS table spaces and DMS table space file containers are treated as files in the database manager's interaction with the operating system, and file handles are required. More files are generally used by SMS table spaces compared to the number of containers used for a DMS file table space. Therefore, if you are using

SMS table spaces, you will need a larger value for this parameter compared to what you would require for DMS file table spaces.

You can also use this parameter to ensure that the overall total of file handles used by the database manager does not exceed the operating system limit by limiting the number of handles per agent to a specific number; the actual number will vary depending on the number of agents running concurrently.

Related reference:

- "maxtotfilop - Maximum total files open " on page 430
- "maxappls - Maximum number of active applications " on page 427
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

maxtotfilop - Maximum total files open

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	16 000 [100 – 32 768]
Unit of Measure	Counter

This parameter defines the maximum number of files that can be opened by all agents and other threads executing in a single database manager instance. If opening a file causes this value to be exceeded, an error is returned to your application.

Note: This parameter does not apply to UNIX-based platforms.

Recommendation: When setting this parameter, you should consider the number of file handles that could be used for each database in the database manager instance. To estimate an upper limit for this parameter:

1. Calculate the maximum number of file handles that could be opened for each database in the instance, using the following formula:

   ```
   maxappls * maxfilop
   ```

2. Calculate the sum of above results and verify that it does not exceed the parameter maximum.

If a new database is created, you should re-evaluate the value for this parameter.

Related reference:

- "maxfilop - Maximum database files open per application " on page 429
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

num_initagents - Initial number of agents in pool

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 0 [0 — *num_poolagents*]

This parameter determines the initial number of idle agents that are created in the agent pool at DB2START time.

Related reference:

- "maxagents - Maximum number of agents " on page 426
- "num_poolagents - Agent pool size " on page 431
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

num_poolagents - Agent pool size

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (*maxagents* / 2) [-1; 0 — *maxagents*]

When the Concentrator is off, that is, when *max_connections* is equal to or less than *max_coordagents*, this parameter determines the maximum size of the idle agent pool. Idle agents can be used as parallel subagents or as coordinator agents. If more agents are created than is indicated by the value of this parameter, they will be terminated when they finish executing their current request, rather than be returned to the pool.

When the Concentrator is on, that is, when *max_connections* is greater than *max_coordagents*, agents will always be returned to the pool, regardless of the value of this parameter. Based on the system load and the time agents remain idle in the pool, agents might terminate themselves, as necessary, to reduce the size of the idle pool to the configured parameter value.

Except when the Concentrator is on, if the value of this parameter is 0, agents will be created as needed, and will terminate once they finish executing their current request.

Recommendation: If you run a decision-support environment in which few applications connect concurrently, set *num_poolagents* to a small value to avoid having an agent pool that is full of idle agents.

If you run a transaction processing environment in which many applications are concurrently connected, increase the value of *num_poolagents* to avoid the costs associated with the frequent creation and termination of agents.

Related reference:

- "num_initagents - Initial number of agents in pool " on page 431
- "maxagents - Maximum number of agents " on page 426
- "max_querydegree - Maximum query degree of parallelism " on page 492
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Stored procedures and user-defined functions

The following parameters can affect fenced stored procedure and user-defined function performance:

- "fenced_pool - Maximum number of fenced processes "
- "keepfenced - Keep fenced process " on page 433
- "num_initfenced - Initial number of fenced processes " on page 434

fenced_pool - Maximum number of fenced processes

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] -1 (0 – *max_coordagents*)

Unit of Measure Counter

For threaded db2fmp processes (processes serving threadsafe stored procedures and UDFs), this parameter represents the number of threads cached in each db2fmp process. For nonthreaded db2fmp processes, this parameter represents the number of processes cached.

Recommendation: If your environment uses fenced stored procedures or user defined functions, then this parameter can be used to ensure that an appropriate number of db2fmp processes are available to process the maximum number of

concurrent stored procedures and UDFs that run on the instance, ensuring that no new fenced mode processes need to be created as part of stored procedure and UDF execution.

If the parameter is set to –1, the maximum number of cached db2fmp processes will be the same as the value set in the *max_coordagents* parameter.

If you find that the default value is not appropriate for your environment because an inappropriate amount of system resource is being given to db2fmp processes and is affecting performance of the database manager, the following might be useful in providing a starting point for tuning this parameter:

```
fenced_pool = # of applications allowed to make stored procedure and
UDF calls at one time
```

If *keepfenced* is set to *yes*, then each db2fmp process that is created in the cache pool will continue to exist and use system resources even after the fenced routine call has been processed and returned to the agent.

If *keepfenced* is set to *no*, then nonthreaded db2fmp processes will terminate when they complete execution, and there is no cache pool. Multithreaded db2fmp processes will continue to exist, but no threads will be pooled in these processes. This means that even when *keepfenced* is set *no* you can have one threaded C db2fmp process and one threaded Java db2fmp process on your system.

In previous versions of DB2, this parameter was known as *maxdari*.

Related reference:

- "maxagents - Maximum number of agents " on page 426
- "keepfenced - Keep fenced process " on page 433
- "num_initfenced - Initial number of fenced processes " on page 434
- "max_coordagents - Maximum number of coordinating agents " on page 425
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

keepfenced - Keep fenced process

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Yes [Yes; No]

This parameter indicates whether or not a fenced mode process is kept after a fenced mode routine call is complete. Fenced mode processes are created as

separate system entities in order to isolate user-written fenced mode code from the database manager agent process. This parameter is only applicable on database servers.

If *keepfenced* is set to *no*, and the routine being executed is not threadsafe, a new fenced mode process is created and destroyed for each fenced mode invocation. If *keepfenced* is set to *no*, and the routine being executed is threadsafe, the fenced mode process persists, but the thread created for the call is terminated. If *keepfenced* is set to *yes*, a fenced mode process or thread is reused for subsequent fenced mode calls. When the database manager is stopped, all outstanding fenced mode processes and threads will be terminated.

Setting this parameter to *yes* will result in additional system resources being consumed by the database manager for each fenced mode process that is activated, up to the value contained in the *fenced_pool* parameter. A new process is only created when no existing fenced mode process is available to process a subsequent fenced routine invocation. This parameter is ignored if *fenced_pool* is set to 0.

Recommendation: In an environment in which the number of fenced mode requests is large relative to the number of non-fenced mode requests, and system resources are not constrained, then this parameter can be set to *yes*. This will improve the fenced mode process performance by avoiding the initial fenced mode process creation overhead since an existing fenced mode process will be used to process the call. In particular, for Java routines, this will save the cost of starting the Java Virtual Machine (JVM), a very significant performance improvement.

For example, in an OLTP debit-credit banking transaction application, the code to perform each transaction could be performed in a stored procedure which executes in a fenced mode process. In this application, the main workload is performed out of fenced mode processes. If this parameter is set to *no*, each transaction incurs the overhead of creating a new fenced mode process, resulting in a significant performance reduction. If, however, this parameter is set to *yes*, each transaction would try to use an existing fenced mode process, which would avoid this overhead.

In previous versions of DB2, this parameter was known as *keepdari*.

Related reference:

- "fenced_pool - Maximum number of fenced processes " on page 432
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

num_initfenced - Initial number of fenced processes

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type	Configurable
Default [Range]	0 [0 — *max_connections* + (*maxagents* - *max_coordagents*)]

This parameter indicates the initial number of nonthreaded, idle db2fmp processes that are created in the db2fmp pool at DB2START time. Setting this parameter will reduce the initial startup time for running non-threadsafe C and Cobol routines. This parameter is ignored if *keepfenced* is not specified.

It is much more important to set *fenced_pool* to an appropriate size for your system than to start up a number of db2fmp processes at DB2START time.

In previous versions of DB2, this parameter was known as *num_initdaris*.

Related reference:

- "fenced_pool - Maximum number of fenced processes " on page 432
- "keepfenced - Keep fenced process " on page 433
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Logging and recovery

Recovering your environment can be very important to prevent the loss of critical data. A number of parameters are available to help you manage your environment and to ensure that you can perform adequate recovery of your data or transactions. These parameters are grouped into the following categories:

- "Database log files"
- "Database log activity" on page 444
- "Recovery" on page 454
- "Distributed unit of work recovery" on page 465

Database log files

The following parameters provide information about number, size and status of the files used for database logging:

- "logfilsiz - Size of log files " on page 436
- "loghead - First active log file " on page 437
- "logpath - Location of log files " on page 437
- "logprimary - Number of primary log files " on page 437
- "logsecond - Number of secondary log files " on page 439
- "max_log - Maximum log per transaction " on page 440
- "mirrorlogpath - Mirror log path " on page 440
- "newlogpath - Change the database log path " on page 442
- "num_log_span - Number log span " on page 443
- "overflowlogpath - Overflow log path " on page 443

logfilsiz - Size of log files

Configuration Type	Database	
Parameter Type	Configurable	
Default [Range]		
	UNIX	1000 [4 — 262 144]
	Windows	1000 [4 — 262 144]
Unit of Measure	Pages (4 KB)	

This parameter defines the size of each primary and secondary log file. The size of these log files limits the number of log records that can be written to them before they become full and a new log file is required.

The use of primary and secondary log files as well as the action taken when a log file becomes full are dependent on the type of logging that is being performed:

- Circular logging

  A primary log file can be reused when the changes recorded in it have been committed. If the log file size is small and applications have processed a large number of changes to the database without committing the changes, a primary log file can quickly become full. If all primary log files become full, the database manager will allocate secondary log files to hold the new log records.
- Log retention logging

  When a primary log file is full, the log is archived and a new primary log file is allocated.

Recommendation: You must balance the size of the log files with the number of primary log files:

- The value of the *logfilsiz* should be increased if the database has a large number of update, delete, or insert transactions running against it which will cause the log file to become full very quickly.

  Note: The upper limit of log file size, combined with the upper limit of the number of log files (*logprimary* + *logsecond*), gives an upper limit of 256 GB of active log space.

  A log file that is too small can affect system performance because of the overhead of archiving old log files, allocating new log files, and waiting for a usable log file.
- The value of the *logfilsiz* should be reduced if disk space is scarce, since primary logs are preallocated at this size.

  A log file that is too large can reduce your flexibility when managing archived log files and copies of log files, since some media might not be able to hold an entire log file.

If you are using log retention, the current active log file is closed and truncated when the last application disconnects from a database. When the next connection to the database occurs, the next log file is used. Therefore, if you understand the logging requirements of your concurrent applications, you might be able to determine a log file size that will not allocate excessive amounts of wasted space.

Related reference:

- "logprimary - Number of primary log files " on page 437

- "logsecond - Number of secondary log files " on page 439
- "softmax - Recovery range and soft checkpoint interval " on page 451
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

loghead - First active log file

Configuration Type Database

Parameter Type Informational

This parameter contains the name of the log file that is currently active.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

logpath - Location of log files

Configuration Type Database

Parameter Type Informational

This parameter contains the current path being used for logging purposes. You cannot change this parameter directly as it is set by the database manager after a change to the *newlogpath* parameter becomes effective.

When a database is created, the recovery log file for it is created in a subdirectory of the directory containing the database. The default is a subdirectory named SQLOGDIR under the directory created for the database.

Related reference:

- "newlogpath - Change the database log path " on page 442
- "GET DATABASE CONFIGURATION command" in *Command Reference*

logprimary - Number of primary log files

Configuration Type Database

Parameter Type Configurable

Default [Range] 3 [2 – 256]

Unit of Measure Counter

When Allocated

- The database is created
- A log is moved to a different location (which occurs when the *logpath* parameter is updated)
- Following a increase in the value of this parameter (*logprimary*), during the next database connection after all users have disconnected
- A log file has been archived and a new log file is allocated (the *logretain* or *userexit* parameter must be enabled)

- If the *logfilsiz* parameter has been changed, the active log files are re-sized during the next database connection after all users have disconnected.

When Freed

Not freed unless this parameter decreases. If decreased, unneeded log files are deleted during the next connection to the database.

The primary log files establish a fixed amount of storage allocated to the recovery log files. This parameter allows you to specify the number of primary log files to be preallocated.

Under circular logging, the primary logs are used repeatedly in sequence. That is, when a log is full, the next primary log in the sequence is used if it is available. A log is considered available if all units of work with log records in it have been committed or rolled-back. If the next primary log in sequence is not available, then a secondary log is allocated and used. Additional secondary logs are allocated and used until the next primary log in the sequence becomes available or the limit imposed by the *logsecond* parameter is reached. These secondary log files are dynamically deallocated as they are no longer needed by the database manager.

The number of primary and secondary log files must comply with the following:

- If *logsecond* has a value of -1, *logprimary* <= 256.
- If *logsecond* does not have a value of -1, (*logprimary* + *logsecond*) <= 256.

Recommendation: The value chosen for this parameter depends on a number of factors, including the type of logging being used, the size of the log files, and the type of processing environment (for example, length of transactions and frequency of commits).

Increasing this value will increase the disk requirements for the logs because the primary log files are preallocated during the very first connection to the database.

If you find that secondary log files are frequently being allocated, you might be able to improve system performance by increasing the log file size (*logfilsiz*) or by increasing the number of primary log files.

For databases that are not frequently accessed, in order to save disk storage, set the parameter to 2. For databases enabled for roll-forward recovery, set the parameter larger to avoid the overhead of allocating new logs almost immediately.

You can use the database system monitor to help you size the primary log files. Observation of the following monitor values over a period of time will aid in better tuning decisions, as average values might be more representative of your ongoing requirements.

- *sec_log_used_top (maximum secondary log space used)*
- *tot_log_used_top (maximum total log space used)*
- *sec_logs_allocated (secondary logs allocated currently)*

Related reference:

- "sec_log_used_top - Maximum Secondary Log Space Used monitor element" in *System Monitor Guide and Reference*
- "tot_log_used_top - Maximum Total Log Space Used monitor element" in *System Monitor Guide and Reference*

- “sec_logs_allocated - Secondary Logs Allocated Currently monitor element” in *System Monitor Guide and Reference*
- “logfilsiz - Size of log files ” on page 436
- “logsecond - Number of secondary log files ” on page 439
- “logretain - Log retain enable ” on page 449
- “userexit - User exit enable ” on page 452
- “GET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION command” in *Command Reference*

logsecond - Number of secondary log files

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	2 [-1; 0 – 254]
Unit of Measure	Counter
When Allocated	As needed when *logprimary* is insufficient (see detail below)
When Freed	Over time as the database manager determines they will no longer be required.

This parameter specifies the number of secondary log files that are created and used for recovery log files (only as needed). When the primary log files become full, the secondary log files (of size *logfilsiz*) are allocated one at a time as needed, up to a maximum number as controlled by this parameter. An error code will be returned to the application, and the database will be shut down, if more secondary log files are required than are allowed by this parameter.

If you set *logsecond* to -1, the database is configured with infinite active log space. There is no limit on the size or the number of in-flight transactions running on the database. If you set *logsecond* to -1, you still use the *logprimary* and *logfilsiz* configuration parameters to specify how many log files DB2 should keep in the active log path. If DB2 needs to read log data from a log file, but the file is not in the active log path, DB2 will invoke the userexit program to retrieve the log file from the archive to the active log path. (DB2 will retrieve the files to the overflow log path, if you have configured one.) Once the log file is retrieved, DB2 will cache this file in the active log path so that other reads of log data from the same file will be fast. DB2 will manage the retrieval, caching, and removal of these log files as required.

If your log path is a raw device, you must configure the *overflowlogpath* configuration parameter in order to set *logsecond* to -1.

By setting *logsecond* to -1, you will have no limit on the size of the unit of work or the number of concurrent units of work. However, rollback (both at the savepoint level and at the unit of work level) could be very slow due to the need to retrieve log files from the archive. Crash recovery could also be very slow for the same reason. DB2 will write a message to the administration notification log to warn you that the current set of active units of work has exceeded the primary log files. This is an indication that rollback or crash recovery could be extremely slow.

To set *logsecond* to -1, the *logarchmeth1* configuration parameter must be set to a value other than OFF or LOGRETAIN.

Recommendation: Use secondary log files for databases that have periodic needs for large amounts of log space. For example, an application that is run once a month might require log space beyond that provided by the primary log files. Since secondary log files do not require permanent file space they are advantageous in this type of situation.

Related reference:

- "logfilsiz - Size of log files " on page 436
- "logprimary - Number of primary log files " on page 437
- "logretain - Log retain enable " on page 449
- "userexit - User exit enable " on page 452
- "overflowlogpath - Overflow log path " on page 443
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

max_log - Maximum log per transaction

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	0 [0 — 100]
Unit of Measure	Percentage

If the value is not 0, this parameter indicates the percentage of primary log space that can be consumed by one transaction.

If the value is set to 0, there is no limit regarding how much space (as a percentage of total active log space) one single transaction can consume. This was the behavior of transactions prior to Version 8.

Related reference:

- "num_log_span - Number log span " on page 443
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

mirrorlogpath - Mirror log path

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any valid path or device]

This parameter allows you to specify a string of up to 242 bytes for the mirror log path. The string must point to a path name, and it must be a fully qualified path name, not a relative path name.

Note: In a single or multi-partition DB2 ESE environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

If *mirrorlogpath* is configured, DB2 will create active log files in both the log path and the mirror log path. All log data will be written to both paths. The mirror log path has a duplicated set of active log files, such that if there is a disk error or human error that destroys active log files on one of the paths, the database can still function.

If the mirror log path is changed, there might be log files in the old mirror log path. These log files might not have been archived, so you might need to archive these log files manually. Also, if you are running replication on this database, replication might still need the log files from before the log path change. If the database is configured with the User Exit Enable (*userexit*) database configuration parameter set to Yes, and if all the log files have been archived either by DB2 automatically or by yourself manually, then DB2 will be able to retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old mirror log path to the new mirror log path.

If *logpath* or *newlogpath* specifies a raw device as the location where the log files are stored, mirror logging, as indicated by *mirrorlogpath*, is not allowed. If *logpath* or *newlogpath* specifies a file path as the location where the log files are stored, mirror logging is allowed and *mirrorlogpath* must also specify a file path.

Recommendation: Just like the log files, the mirror log files should be on a physical disk that does not have high I/O.

It is strongly recommended that this path be on a separate device than the primary log path.

You can use the database system monitor to track the number of I/Os related to database logging.

The following data elements return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

- *log_reads* (number of log pages read)
- *log_writes* (number of log pages written).

Related reference:

- "newlogpath - Change the database log path " on page 442
- "logpath - Location of log files " on page 437
- "log_reads - Number of Log Pages Read monitor element" in *System Monitor Guide and Reference*
- "log_writes - Number of Log Pages Written monitor element" in *System Monitor Guide and Reference*
- "overflowlogpath - Overflow log path " on page 443
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

newlogpath - Change the database log path

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	Null [any valid path or device]

This parameter allows you to specify a string of up to 242 bytes to change the location where the log files are stored. The string can point to either a path name or to a raw device. Note that as of DB2 Version 9, the use of raw devices for database logging is deprecated. As an alternative to using raw logs, you can use either direct input/output (DIO) or concurrent input/output (CIO).

If the string points to a path name, it must be a fully qualified path name, not a relative path name.

In a single or multi-partition DB2 ESE environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

If you want to use replication, and your log path is a raw device, the *overflowlogpath* configuration parameter must be configured.

To specify a device, specify a string that the operating system identifies as a device. For example:

- On Windows, `\\.\d:` or `\\.\PhysicalDisk5`

  Note: You must have Windows Version 4.0 with Service Pack 3 or later installed to be able to write logs to a device.
- On UNIX-based platforms, `/dev/rdblog8`

Note: You can only specify a device on AIX, Windows 2000, Windows, Solaris Operating Environment, HP-UX, and Linux platforms.

The new setting does not become the value of *logpath* until both of the following occur:

- The database is in a consistent state, as indicated by the *database_consistent* parameter.
- All users are disconnected from the database

When the first new connection is made to the database, the database manager will move the logs to the new location specified by *logpath*.

There might be log files in the old log path. These log files might not have been archived. You might need to archive these log files manually. Also, if you are running replication on this database, replication might still need the log files from before the log path change. If the database is configured with the User Exit Enable (*userexit*) database configuration parameter set to Yes, and if all the log files have been archived either by DB2 automatically or by yourself manually, then DB2 will be able to retrieve the log files to complete the replication process. Otherwise, you can copy the files from the old log path to the new log path.

If *logpath* or *newlogpath* specifies a raw device as the location where the log files are stored, mirror logging, as indicated by *mirrorlogpath*, is not allowed. If *logpath* or

newlogpath specifies a file path as the location where the log files are stored, mirror logging is allowed and *mirrorlogpath* must also specify a file path.

Recommendation: Ideally, the log files will be on a physical disk which does **not** have high I/O. For instance, avoid putting the logs on the same disk as the operating system or high volume databases. This will allow for efficient logging activity with a minimum of overhead such as waiting for I/O.

You can use the database system monitor to track the number of I/Os related to database logging.

The monitor elements *log_reads* (number of log pages read) and *log_writes* (number of log pages written) return the amount of I/O activity related to database logging. You can use an operating system monitor tool to collect information about other disk I/O activity, then compare the two types of I/O activity.

Related reference:

- "log_reads - Number of Log Pages Read monitor element" in *System Monitor Guide and Reference*
- "log_writes - Number of Log Pages Written monitor element" in *System Monitor Guide and Reference*
- "logpath - Location of log files " on page 437
- "database_consistent - Database is consistent " on page 473
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

num_log_span - Number log span

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	0 [0 — 65 535]
Unit of Measure	Counter

If the value is not 0, this parameter indicates the number of active log files that one active transaction is allowed to span.

If the value is set to 0, there is no limit to how many log files one single transaction can span. This was the behavior of transactions prior to Version 8.

Related reference:

- "max_log - Maximum log per transaction " on page 440
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

overflowlogpath - Overflow log path

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate

Default [Range] Null [any valid path]

This parameter can be used for several functions, depending on your logging requirements.

- This parameter allows you to specify a location for DB2 to find log files that are needed for a rollforward operation. It is similar to the OVERFLOW LOG PATH option on the ROLLFORWARD command. Instead of always specifying OVERFLOW LOG PATH on every ROLLFORWARD command, you can set this configuration parameter once. However, if both are used, the OVERFLOW LOG PATH option will overwrite the *overflowlogpath* configuration parameter, for that particular rollforward operation.
- If *logsecond* is set to -1, *overflowlogpath* allows you to specify a directory for DB2 to store active log files retrieved from the archive. (Active log files have to be retrieved for rollback operations if they are no longer in the active log path). Without overflowlogpath, DB2 will retrieve the log files into the active log path. Using *overflowlogpath* allows you to provide additional resource for DB2 to store the retrieved log files. The benefit includes spreading the I/O cost to different disks, and allowing more log files to be stored in the active log path.
- If you need to use the db2ReadLog API (prior to DB2 V8, db2ReadLog was called sqlurlog) for replication, for example, *overflowlogpath* allows you to specify a location for DB2 to search for log files that are needed for this API. If the log file is not found (in either the active log path or the overflow log path) and the database is configured with *userexit* enabled, DB2 will retrieve the log file. *overflowlogpath* also allows you to specify a directory for DB2 to store the log files retrieved. The benefit comes from reducing the I/O cost on the active log path and allowing more log files to be stored in the active log path.
- If you have configured a raw device for the active log path, *overflowlogpath* must be configured if you want to set *logsecond* to -1, or if you want to use the db2ReadLog API.

To set *overflowlogpath*, specify a string of up to 242 bytes. The string must point to a path name, and it must be a fully qualified path name, not a relative path name. The path name must be a directory, not a raw device.

Note: In a single or multi-partition DB2 ESE environment, the node number is automatically appended to the path. This is done to maintain the uniqueness of the path in multiple logical node configurations.

Related reference:

- "logsecond - Number of secondary log files " on page 439
- "ROLLFORWARD DATABASE command" in *Command Reference*
- "db2ReadLog API - Extracts log records" in *Administrative API Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Database log activity

The following parameters can influence the type and performance of database logging:

- "archretrydelay - Archive retry delay on error " on page 445
- "blk_log_dsk_ful - Block on log disk full " on page 445
- "failarchpath - Failover log archive path " on page 446

archretrydelay - Archive retry delay on error

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] 20 [0 – 65 535]

This parameter specifies the number of seconds to wait after a failed archive attempt before trying to archive the log file again. Subsequent retries will only take affect if the value of the *numarchretry* database configuration parameter is at least 1.

Related reference:

- “GET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION command” in *Command Reference*

blk_log_dsk_ful - Block on log disk full

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] No [Yes; No]

This configuration parameter can be set to prevent disk full errors from being generated when DB2 cannot create a new log file in the active log path. Instead, DB2 will attempt to create the log file every five minutes until it succeeds. After each attempt, DB2 writes a message to the Administration Notification log. The only way that you can confirm that your application is hanging because of a log disk full condition is to monitor the Administration Notification log. Until the log file is successfully created, any user application that attempts to update table data will not be able to commit transactions. Read-only queries might not be directly affected; however, if a query needs to access data that is locked by an update

request, or a data page that is fixed in the buffer pool by the updating application, read-only queries will also appear to hang.

Setting *blk_log_dsk_ful* to *yes* causes applications to hang when DB2 encounters a log disk full error, thus allowing you to resolve the error and allowing the transaction to complete. You can resolve a disk full situation by moving old log files to another file system or by enlarging the file system, so that hanging applications can complete.

If *blk_log_dsk_ful* is set to *no*, then a transaction that receives a log disk full error will fail and will be rolled back. In some situations, the database will come down if a transaction causes a log disk full error.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

failarchpath - Failover log archive path

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Null [ ]

This parameter specifies a path to which DB2 will try to archive log files if the log files cannot be archived to either the primary or the secondary (if set) archive destinations because of a media problem affecting those destinations. This specified path must reference a disk.

If there are log files in the path specified by the current value of *failarchpath*, any updates to *failarchpath* will not take effect immediately. Instead, the update will take effect when all applications disconnect.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logarchmeth1 - Primary log archive method

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Off [ ]

This parameter specifies the media type of the primary destination for archived logs.

Related concepts:

- "Cross-node recovery with the db2adutl command and the logarchopt1 and vendoropt database configuration parameters" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logarchmeth2 - Secondary log archive method

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Off [ ]

This parameter specifies the media type of the secondary destination for archived logs. If this path is specified, log files will be archived to both this destination and the destination specified by the *logarchmeth1* database configuration parameter.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logarchopt1 - Primary log archive options

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Null [ ]

This parameter specifies the options field for the primary destination for archived logs (if required).

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logarchopt2 - Secondary log archive options

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Null [ ]

This parameter specifies the options field for the secondary destination for archived logs (if required).

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logindexbuild - Log index pages created

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Off [On; Off]

This parameter specifies whether index creation, recreation, or reorganization operations are to be logged so that indexes can be reconstructed during DB2 rollforward operations or high availability disaster recovery (HADR) log replay procedures.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

logretain - Log retain enable

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [Recovery; No]

The values are as follows:

- No, to indicate that logs are not retained.
- Recovery, to indicate that the logs are retained, and can be used for forward recovery.

If *logretain* is set to Recovery or *userexit* is set to Yes, the active log files will be retained and become online archive log files for use in roll-forward recovery. This is called log retention logging.

After *logretain* is set to Recovery or *userexit* is set to Yes (or both), you must make a full backup of the database. This state is indicated by the *backup_pending* flag parameter.

If *logretain* is set to No and *userexit* is set to No, roll-forward recovery is not available for the database because logs are not retained. In this situation, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

Related reference:

- "userexit - User exit enable " on page 452
- "log_retain_status - Log retain status indicator " on page 473
- "backup_pending - Backup pending indicator " on page 473
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

mincommit - Number of commits to group

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	1 [1 – 25]
Unit of Measure	Counter

This parameter allows you to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help reduce the database manager overhead associated with writing log records. As a result, this will improve performance when you have multiple applications running against a database and many commits are requested by the applications within a very short time frame.

This grouping of commits will only occur when the value of this parameter is greater than one and when the number of applications connected to the database is greater than or equal to the value of this parameter. When commit grouping is

being performed, application commit requests could be held until either one second has elapsed or the number of commit requests equals the value of this parameter.

This parameter should be incremented by small amounts only; for example one (1). You should also use multi-user tests to verify that increasing the value of this parameter provides the expected results.

Changes to the value specified for this parameter take effect immediately; you do not have to wait until all applications disconnect from the database.

Recommendation: Increase this parameter from its default value if multiple read/write applications typically request concurrent database commits. This will result in more efficient logging file I/O as it will occur less frequently and write more log records each time it does occur.

You could also sample the number of transactions per second and adjust this parameter to accommodate the peak number of transactions per second (or some large percentage of it). Accommodating peak activity would minimize the overhead of writing log records during transaction intensive periods.

If you increase *mincommit*, you might also need to increase the *logbufsz* parameter to avoid having a full log buffer force a write during these transaction intensive periods. In this case, the *logbufsz* should be equal to:

```
mincommit * (log space used, on average, by a transaction)
```

You can use the database system monitor to help you tune this parameter in the following ways:

- Calculating the peak number of transactions per second:

  Taking monitor samples throughout a typical day, you can determine your transaction intensive periods. You can calculate the total transactions by adding the following monitor elements:

  – *commit_sql_stmts (commit statements attempted)*
  – *rollback_sql_stmts (rollback statements attempted)*

  Using this information and the available timestamps, you can calculate the number of transactions per second.
- Calculating the log space used per transaction:

  Using sampling techniques over a period of time and a number of transactions, you can calculate an average of the log space used with the following monitor element:

  – *log_space_used (unit of work log space used)*

Related reference:

- "commit_sql_stmts - Commit Statements Attempted monitor element" in *System Monitor Guide and Reference*
- "rollback_sql_stmts - Rollback Statements Attempted monitor element" in *System Monitor Guide and Reference*
- "uow_log_space_used - Unit of Work Log Space Used monitor element" in *System Monitor Guide and Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

numarchretry - Number of retries on error

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] 5 [0 – 65 535]

This parameter specifies the number of times that DB2 is to try archiving a log file to the primary or the secondary archive directory before trying to archive log files to the failover directory. This parameter is only used if the *failarchpath* database configuration parameter is set. If *numarchretry* is not set, DB2 will continuously retry archiving to the primary or the secondary log path.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

softmax - Recovery range and soft checkpoint interval

Configuration Type Database

Parameter Type Configurable

Default [Range] 100 [1 – 100 * *logprimary*]

Unit of Measure Percentage of the size of one primary log file

This parameter is used to:

- Influence the number of logs that need to be recovered following a crash (such as a power failure). For example, if the default value is used, the database manager will try to keep the number of logs that need to be recovered to 1. If you specify 300 as the value of this parameter, the database manager will try to keep the number of logs that need to be recovered to 3.

  To influence the number of logs required for crash recovery, the database manager uses this parameter to trigger the page cleaners to ensure that pages older than the specified recovery window are already written to disk.
- Determine the frequency of soft checkpoints.

At the time of a database failure resulting from an event such as a power failure, there might have been changes to the database which:

- Have not been committed, but updated the data in the buffer pool
- Have been committed, but have not been written from the buffer pool to the disk
- Have been committed and written from the buffer pool to the disk.

When a database is restarted, the log files will be used to perform a crash recovery of the database which ensures that the database is left in a consistent state (that is,

all committed transactions are applied to the database and all uncommitted transactions are not applied to the database).

To determine which records from the log file need to be applied to the database, the database manager uses a log control file. This log control file is periodically written to disk, and, depending on the frequency of this event, the database manager might be applying log records of committed transactions or applying log records that describe changes that have already been written from the buffer pool to disk. These log records have no impact on the database, but applying them introduces some overhead into the database restart process.

The log control file is always written to disk when a log file is full, and during soft checkpoints. You can use this configuration parameter to trigger additional soft checkpoints.

The timing of soft checkpoints is based on the difference between the "current state" and the "recorded state", given as a percentage of the *logfilsiz*. The "recorded state" is determined by the oldest valid log record indicated in the log control file on disk, while the "current state" is determined by the log control information in memory. (The oldest valid log record is the first log record that the recovery process would read.) The soft checkpoint will be taken if the value calculated by the following formula is greater than or equal to the value of this parameter:

```
( (space between recorded and current states) / logfilsiz ) * 100
```

Recommendation: You might want to increase or reduce the value of this parameter, depending on whether your acceptable recovery window is greater than or less than one log file. Lowering the value of this parameter will cause the database manager both to trigger the page cleaners more often and to take more frequent soft checkpoints. These actions can reduce both the number of log records that need to be processed and the number of redundant log records that are processed during crash recovery.

Note however, that more page cleaner triggers and more frequent soft checkpoints increase the overhead associated with database logging, which can impact the performance of the database manager. Also, more frequent soft checkpoints might not reduce the time required to restart a database, if you have:

- Very long transactions with few commit points.
- A very large buffer pool and the pages containing the committed transactions are not written back to disk very frequently. (Note that the use of asynchronous page cleaners can help avoid this situation.)

In both of these cases, the log control information kept in memory does not change frequently and there is no advantage in writing the log control information to disk, unless it has changed.

Related reference:

- "logfilsiz - Size of log files " on page 436
- "logprimary - Number of primary log files " on page 437
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

userexit - User exit enable

Configuration Type Database

Parameter Type Configurable

Default [Range] Off[On; Off]

If this parameter is enabled, log retention logging is performed regardless of how the *logretain* parameter is set. This parameter also indicates that a user exit program should be used to archive and retrieve the log files. Log files are archived when the log file is full. They are retrieved when the ROLLFORWARD utility needs to use them to restore a database.

After *logretain*, or *userexit*, or both of these parameters are enabled, you must make a full backup of the database. This state is indicated by the *backup_pending* flag parameter.

If both of these parameters are de-selected, roll-forward recovery becomes unavailable for the database because logs will no longer be retained. In this case, the database manager deletes all log files in the *logpath* directory (including online archive log files), allocates new active log files, and reverts to circular logging.

Related reference:

- "ROLLFORWARD DATABASE command" in *Command Reference*
- "User exit for database recovery" in *Data Recovery and High Availability Guide and Reference*
- "logretain - Log retain enable " on page 449
- "user_exit_status - User exit status indicator " on page 475
- "backup_pending - Backup pending indicator " on page 473
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

vendoropt - Vendor options

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Default [Range] Null [ ]

This parameter specifies additional parameters that DB2 might need to use to communicate with storage systems during backup, restore, or load copy operations.

Related concepts:

- "Cross-node recovery with the db2adutl command and the logarchopt1 and vendoropt database configuration parameters" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Recovery

The following parameters affect various aspects of database recovery:

- "autorestart - Auto restart enable "
- "dft_loadrec_ses - Default number of load recovery sessions " on page 455
- "indexrec - Index re-creation time " on page 459
- "num_db_backups - Number of database backups " on page 461
- "rec_his_retentn - Recovery history retention period " on page 462
- "trackmod - Track modified pages enable " on page 463

See also "Distributed unit of work recovery" on page 465.

The following parameters are used when working with Tivoli Storage Manager (TSM):

- "tsm_mgmtclass - Tivoli Storage Manager management class " on page 463
- "tsm_nodename - Tivoli Storage Manager node name " on page 464
- "tsm_owner - Tivoli Storage Manager owner name " on page 464
- "tsm_password - Tivoli Storage Manager password " on page 464

The following parameters pertain to high availability disaster recovery (HADR):

- "hadr_db_role - HADR database role " on page 455
- "hadr_local_host - HADR local host name " on page 456
- "hadr_local_svc - HADR local service name " on page 456
- "hadr_remote_host - HADR remote host name " on page 457
- "hadr_remote_inst - HADR instance name of the remote server " on page 457
- "hadr_remote_svc - HADR remote service name " on page 458
- "hadr_syncmode - HADR synchronization mode for log write in peer state " on page 458
- "hadr_timeout - HADR timeout value " on page 459

autorestart - Auto restart enable

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	On [On; Off]

When this parameter is set on, the database manager automatically calls the restart database utility, if needed, when an application connects to a database. *Crash recovery* is the operation performed by the restart database utility. It is performed if the database terminated abnormally while applications were connected to it. An abnormal termination of the database could be caused by a power failure or a system software failure. It applies any committed transactions that were in the database buffer pool but were not written to disk at the time of the failure. It also backs out any uncommitted transactions that might have been written to disk.

If *autorestart* is not enabled, then an application that attempts to connect to a database which needs to have crash recovery performed (needs to be restarted) will receive a SQL1015N error. In this case, the application can call the restart database utility, or you can restart the database by selecting the restart operation of the recovery tool.

Related concepts:

- "Crash recovery" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "RESTART DATABASE command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

dft_loadrec_ses - Default number of load recovery sessions

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	1 [1 – 30 000]
Unit of Measure	Counter

This parameter specifies the default number of sessions that will be used during the recovery of a table load. The value should be set to an optimal number of I/O sessions to be used to retrieve a load copy. The retrieval of a load copy is an operation similar to restore. You can override this parameter through entries in the copy location file specified by the environment variable DB2LOADREC.

The default number of buffers used for load retrieval is two more than the value of this parameter. You can also override the number of buffers in the copy location file.

This parameter is applicable only if roll forward recovery is enabled.

Related concepts:

- "Load overview" in *Data Movement Utilities Guide and Reference*

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_db_role - HADR database role

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Informational

This parameter indicates the current role of a database, whether the database is online or offline. Valid values are: STANDARD, PRIMARY, or STANDBY.

Note: Although the GET SNAPSHOT FOR DATABASE command returns high availability disaster recovery (HADR) status, it does so only when the database is online.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

hadr_local_host - HADR local host name

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter specifies the local host for high availability disaster recovery (HADR) TCP communication. Either a host name or an IP address can be used. If a host name is specified and it maps to multiple IP addresses, an error is returned, and HADR will not start up. If the host name maps to multiple IP addresses (even if you specify the same host name on primary and standby), primary and standby can end up mapping this host name to different IP addresses, because some DNS servers return IP address lists in non-deterministic order.

A host name is in the form: myserver.ibm.com. An IP address is in the form: ″12.34.56.78″.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_local_svc - HADR local service name

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter specifies the TCP service name or port number for which the local high availability disaster recovery (HADR) process accepts connections.

The value for hadr_local_svc on the Primary or Standby database systems cannot be the same as the value of svcename or svcename +1 on their respective nodes.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_remote_host - HADR remote host name

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter specifies the TCP/IP host name or IP address of the remote high availability disaster recovery (HADR) node. Similar to *hadr_local_host*, this parameter must map to only one IP address.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_remote_inst - HADR instance name of the remote server

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter specifies the instance name of the remote server. Administration tools, such as the DB2 Control Center, use this parameter to contact the remote server. High availability disaster recovery (HADR) also checks whether a remote database requesting a connection belongs to the declared remote instance.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_remote_svc - HADR remote service name

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter specifies the TCP service name or port number that will be used by the remote high availability disaster recovery (HADR) node.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_syncmode - HADR synchronization mode for log write in peer state

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] NEARSYNC [ASYNC; SYNC]

This parameter specifies the synchronization mode, which determines how primary log writes are synchronized with the standby when the systems are in peer state. Valid values are:

SYNC This mode provides the greatest protection against transaction loss, but at a higher cost of transaction response time.

NEARSYNC This mode provides somewhat less protection against transaction loss, in exchange for a shorter transaction response time than that of SYNC mode.

ASYNC This mode has the highest probability of transaction loss in the event of primary failure, in exchange for the shortest transaction response time among the three modes.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

hadr_timeout - HADR timeout value

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 120 [1 – 4 294 967 295]

This parameter specifies the time (in seconds) that the high availability disaster recovery (HADR) process waits before considering a communication attempt to have failed.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

indexrec - Index re-creation time

Configuration Type Database and Database Manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

UNIX Database Manager
restart [restart; restart_no_redo; access; access_no_redo]

Windows Database Manager
restart [restart; restart_no_redo; access; access_no_redo]

Database Use system setting [system; restart; restart_no_redo; access; access_no_redo]

This parameter indicates when the database manager will attempt to rebuild invalid indexes, and whether or not any index build will be redone during DB2 rollforward or HADR log replay on the standby database. There are five possible settings for this parameter:

SYSTEM *use system setting* specified in the database manager configuration file to decide when invalid indexes will be rebuilt, and whether any index build log records are to be redone during DB2 rollforward or HADR log replay. (Note: This setting is only valid for database configurations.)

ACCESS Invalid indexes are rebuilt when the index is first accessed. Any fully logged index builds are redone during DB2 rollforward or HADR log replay. When HADR is started and an HADR takeover occurs, any invalid indexes are rebuilt after takeover when the underlying table is first accessed.

ACCESS_NO_REDO
Invalid indexes will be rebuilt when the underlying table is first accessed. Any fully logged index build will not be redone during DB2 rollforward or HADR log replay and those indexes will be left invalid. When HADR is started and an HADR takeover takes place, any invalid indexes will be rebuilt after takeover when the underlying table is first accessed.

RESTART The default value for *indexrec*. Invalid indexes will be rebuilt when a RESTART DATABASE command is either explicitly or implicitly issued. Any fully logged index build will be redone during DB2 rollforward or HADR log replay. When HADR is started and an HADR takeover takes place, any invalid indexes will be rebuilt at the end of takeover.

Note that a RESTART DATABASE command is implicitly issued if the *autorestart* parameter is enabled.

RESTART_NO_REDO
Invalid indexes will be rebuilt when a RESTART DATABASE command is either explicitly or implicitly issued. (A RESTART DATABASE command is implicitly issued if the *autorestart* parameter is enabled.) Any fully logged index build will not be redone during DB2 rollforward or HADR log replay and instead those indexes will be rebuilt when rollforward completes or when HADR takeover takes place.

Indexes can become invalid when fatal disk problems occur. If this happens to the data itself, the data could be lost. However, if this happens to an index, the index can be recovered by re-creating it. If an index is rebuilt while users are connected to the database, two problems could occur:

- An unexpected degradation in response time might occur as the index file is re-created. Users accessing the table and using this particular index would wait while the index was being rebuilt.
- Unexpected locks might be held after index re-creation, especially if the user transaction that caused the index to be re-created never performed a COMMIT or ROLLBACK.

Recommendation: The best choice for this option on a high-user server and if restart time is not a concern, would be to have the index rebuilt at DATABASE RESTART time as part of the process of bringing the database back online after a crash.

Setting this parameter to "ACCESS" or to "ACCESS_NO_REDO" will result in a degradation of the performance of the database manager while the index is being re-created. Any user accessing that specific index or table would have to wait until the index is recreated.

If this parameter is set to "RESTART", the time taken to restart the database will be longer due to index re-creation, but normal processing would not be impacted once the database has been brought back online.

Note: At database recovery time, all SQL procedure executables on the file system that belong to the database being recovered are removed. If *indexrec* is set to RESTART, all SQL procedure executables are extracted from the database catalog and put back on the file system at the next connection to the database. If *indexrec* is not set to RESTART, an SQL executable is extracted to the file system only on first execution of that SQL procedure.

The difference between the RESTART and the RESTART_NO_REDO values, or between the ACCESS and the ACCESS_NO_REDO values, is only significant when full logging is activated for index build operations, such as CREATE INDEX and REORG INDEX operations, or for an index rebuild. You can activate logging by enabling the *logindexbuild* database configuration parameter or by enabling LOG INDEX BUILD when altering a table. By setting *indexrec* to either RESTART or ACCESS, operations involving a logged index build can be rolled forward without leaving the index object in an invalid state, which would require the index to be rebuilt at a later time.

Related tasks:

- "Backing up and restoring SQL procedures created prior to DB2 8.2" in *SQL Guide*

Related reference:

- "RESTART DATABASE command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "autorestart - Auto restart enable " on page 454
- "logindexbuild - Log index pages created " on page 448
- "ALTER TABLE statement" in *SQL Reference, Volume 2*

num_db_backups - Number of database backups

Configuration Type Database

Parameter Type	Configurable online
Propagation Class	Transaction boundary
Default [Range]	12 [1 — 32 767]

This parameter specifies the number of database backups to retain for a database. After the specified number of backups is reached, old backups are marked as expired in the recovery history file. Recovery history file entries for the table space backups and load copy backups that are related to the expired database backup are also marked as expired. When a backup is marked as expired, the physical backups can be removed from where they are stored (for example, disk, tape, TSM). The next database backup will prune the expired entries from the recovery history file.

When a database backup is marked as expired in the history file, any corresponding file backups linked through a DB2 Data Links Manager will be removed from its archive server.

The *rec_his_retentn* configuration parameter should be set to a value compatible with the value of *num_db_backups*. For example, if *num_db_backup* is set to a large value, the value for *rec_his_retentn* should be large enough to support that number of backups.

Related reference:

- "rec_his_retentn - Recovery history retention period " on page 462
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*

rec_his_retentn - Recovery history retention period

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	366 [-1; 0 — 30 000]
Unit of Measure	Days

This parameter is used to specify the number of days that historical information on backups should be retained. If the recovery history file is not needed to keep track of backups, restores, and loads, this parameter can be set to a small number.

If value of this parameter is -1, the number of entries indicating full database backups (and any table space backups that are associated with the database backup) will correspond with the value specified by the *num_db_backups* parameter. Other entries in the recovery history file can only be pruned by explicitly using the available commands or APIs.

No matter how small the retention period, the most recent full database backup plus its restore set will always be kept, unless you use the PRUNE utility with the FORCE option.

Related reference:

- "PRUNE HISTORY/LOGFILE command" in *Command Reference*
- "num_db_backups - Number of database backups " on page 461

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

trackmod - Track modified pages enable

Configuration Type Database

Parameter Type Configurable

Default [Range] No [Yes, No]

When this parameter is set to ″Yes″, the database manager tracks database modifications so that the backup utility can detect which subsets of the database pages must be examined by an incremental backup and potentially included in the backup image. After setting this parameter to ″Yes″, you must take a full database backup in order to have a baseline against which incremental backups can be taken. Also, if this parameter is enabled and if a table space is created, then a backup must be taken which contains that table space. This backup could be either a database backup or a table space backup. Following the backup, incremental backups will be permitted to contain this table space.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

tsm_mgmtclass - Tivoli Storage Manager management class

Configuration Type Database

Parameter Type Configurable

Default [Range] Null [any string]

The Tivoli Storage Manager management class determines how the TSM server should manage the backup versions of the objects being backed up.

The default is that there is no DB2-specified management class.

When performing any TSM backup, before using the management class specified by the database configuration parameter, TSM first attempts to bind the backup object to the management class specified in the INCLUDE-EXCLUDE list found in the TSM client options file. If a match is not found, the default TSM management class specified on the TSM server will be used. TSM will then rebind the backup object to the management class specified by the database configuration parameter.

Thus, the default management class, as well as the management class specified by the database configuration parameter, must contain a backup copy group, or the backup operation will fail.

Related reference:

- "Tivoli Storage Manager" in *Data Recovery and High Availability Guide and Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

tsm_nodename - Tivoli Storage Manager node name

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Statement boundary
Default [Range]	Null [any string]

This parameter is used to override the default setting for the node name associated with the Tivoli Storage Manager (TSM) product. The node name is needed to allow you to restore a database that was backed up to TSM from another node.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the *tsm_nodename* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

tsm_owner - Tivoli Storage Manager owner name

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Statement boundary
Default [Range]	Null [any string]

This parameter is used to override the default setting for the owner associated with the Tivoli Storage Manager (TSM) product. The owner name is needed to allow you to restore a database that was backed up to TSM from another node. It is possible for the *tsm_owner* to be overridden during a backup done through DB2 (for example, with the BACKUP DATABASE command).

Note: The owner name is case sensitive.

The default is that you can only restore a database from TSM on the same node from which you did the backup.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

tsm_password - Tivoli Storage Manager password

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Statement boundary
Default [Range]	Null [any string]

This parameter is used to override the default setting for the password associated with the Tivoli Storage Manager (TSM) product. The password is needed to allow you to restore a database that was backed up to TSM from another node.

Note: If the *tsm_nodename* is overridden during a backup done with DB2 (for example, with the BACKUP DATABASE command), the *tsm_password* might also have to be set.

The default is that you can only restore a database from TSM on the same node from which you did the backup. It is possible for the *tsm_nodename* to be overridden during a backup done with DB2.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Distributed unit of work recovery

The following parameters affect the recovery of distributed unit of work (DUOW) transactions:

- "resync_interval - Transaction resync interval "
- "spm_log_file_sz - Sync point manager log file size " on page 466
- "spm_log_path - Sync point manager log file path " on page 467
- "spm_max_resync - Sync point manager resync agent limit " on page 467
- "spm_name - Sync point manager name " on page 467
- "tm_database - Transaction manager database name " on page 468

resync_interval - Transaction resync interval

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	180 [1 – 60 000]
Unit of Measure	Seconds

This parameter specifies the time interval in seconds for which a transaction manager (TM), resource manager (RM) or sync point manager (SPM) should retry the recovery of any outstanding indoubt transactions found in the TM, the RM, or the SPM. This parameter is applicable when you have transactions running in a distributed unit of work (DUOW) environment. This parameter also applies to recovery of federated database systems.

Recommendation: If, in your environment, indoubt transactions will not interfere with other transactions against your database, you might wish to increase the value of this parameter. If you are using a DB2 Connect gateway to access DRDA2 application servers, you should consider the effect indoubt transactions might have

at the application servers even though there will be no interference with local data access. If there are no indoubt transactions, the performance impact will be minimal.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

spm_log_file_sz - Sync point manager log file size

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	256 [4 — 1 000]
Unit of Measure	Pages (4 KB)

This parameter identifies the sync point manager (SPM) log file size in 4 KB pages. The log file is contained in the `spmlog` sub-directory under `sqllib` and is created the first time SPM is started.

Recommendation: The sync point manager log file size should be large enough to maintain performance, but small enough to prevent wasted space. The size required depends on the number of transactions using protected conversations, and how often COMMIT or ROLLBACK is issued.

To change the size of the SPM log file:

1. Determine that there are no indoubt transactions by using the LIST DRDA INDOUBT TRANSACTIONS command.
2. If there are none, stop the database manager.
3. Update the database manager configuration with a new SPM log file size.
4. Go to the `$HOME/sqllib` directory and issue `rm -fr spmlog` to delete the current SPM log. (Note: This shows the AIX command. Other systems might require a different remove or delete command.)
5. Start the database manager. A new SPM log of the specified size is created during the startup of the database manager.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

spm_log_path - Sync point manager log file path

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] sqllib/spmlog [any valid path or device]

This parameter specifies the directory where the sync point manager (SPM) logs are written. By default, the logs are written to the `sqllib/spmlog` directory, which, in a high-volume transaction environment, can cause an I/O bottleneck. Use this parameter to have the SPM log files placed on a faster disk than the current `sqllib/spmlog` directory. This allows for better concurrency among the SPM agents.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

spm_max_resync - Sync point manager resync agent limit

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 20 [10 — 256]

This parameter identifies the number of agents that can simultaneously perform resync operations.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

spm_name - Sync point manager name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Derived from the TCP/IP hostname

This parameter identifies the name of the sync point manager (SPM) instance to the database manager.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

tm_database - Transaction manager database name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] 1ST_CONN [any valid database name]

This parameter identifies the name of the transaction manager (TM) database for each DB2 instance. A TM database can be:

- A local DB2 database
- A remote DB2 database that does not reside on a host or AS/400 system
- A DB2 for OS/390 Version 5 database if accessed via TCP/IP and the sync point manager (SPM) is not used.

The TM database is a database that is used as a logger and coordinator, and is used to perform recovery for indoubt transactions.

You can set this parameter to **1ST_CONN**, which will set the TM database to be the first database to which a user connects.

Recommendation: For simplified administration and operation, you might wish to create a few databases over a number of instances and use these databases exclusively as TM databases.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Database management

A number of parameters provide information about your database or influence the management of your database. These are grouped as follows:

- "Query Enabler"
- "Attributes"
- "Status" on page 473
- "Compiler settings" on page 475
- "Automated maintenance" on page 481

Query Enabler

The following parameter provides information for the control of Query Enabler:

- "dyn_query_mgmt - Dynamic SQL and XQuery query management "

dyn_query_mgmt - Dynamic SQL and XQuery query management

Configuration Type	Database
Parameter Type	Configurable Online
Default [Range]	0 (DISABLE) [1(ENABLE), 0 (DISABLE)]

This parameter is relevant where DB2 Query Patroller is installed. If this parameter is set to "ENABLE", Query Patroller captures information about the query, such as the submitter ID and the estimated cost of execution, as calculated by the optimizer. These values are used to determine whether the query should be managed by Query Patroller, based on user- and system-level thresholds.

If this parameter is set to "DISABLE", Query Patroller does not capture any information about submitted queries, and no query management takes place.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Attributes

The following parameters provide general information about the database:

- "alt_collate - Alternate collating sequence " on page 470
- "codepage - Code page for the database " on page 470
- "codeset - Codeset for the database " on page 470
- "collate_info - Collating information " on page 471
- "country/region - Database territory code " on page 471
- "database_level - Database release level " on page 472
- "pagesize - Database default page size" on page 472
- "release - Configuration file release level " on page 472

- "territory - Database territory " on page 472

With the exception of *alt_collate*, these parameters are provided for informational purposes only.

alt_collate - Alternate collating sequence

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [IDENTITY_16BIT]

This parameter specifies the collating sequence that is to be used for Unicode tables in a non-Unicode database. Until this parameter is set, Unicode tables and routines cannot be created in a non-Unicode database. Once set, this parameter cannot be changed or reset.

This parameter cannot be set for Unicode databases.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

codepage - Code page for the database

Configuration Type Database

Parameter Type Informational

This parameter shows the code page that was used to create the database. The *codepage* parameter is derived based on the *codeset* parameter.

Related reference:

- "codeset - Codeset for the database " on page 470
- "GET DATABASE CONFIGURATION command" in *Command Reference*

codeset - Codeset for the database

Configuration Type Database

Parameter Type Informational

This parameter shows the codeset that was used to create the database. Codeset is used by the database manager to determine *codepage* parameter values.

Related reference:

- "codepage - Code page for the database " on page 470
- "GET DATABASE CONFIGURATION command" in *Command Reference*

collate_info - Collating information

This parameter can only be displayed using the db2CfgGet API. It **cannot** be displayed through the command line processor or the Control Center.

Configuration Type Database

Parameter Type Informational

This parameter provides 260 bytes of database collating information. The first 256 bytes specify the database collating sequence, where byte "n" contains the sort weight of the code point whose underlying decimal representation is "n" in the code page of the database.

The last 4 bytes contain internal information about the type of the collating sequence. The last four bytes of the parameter is an integer. The integer is sensitive to the endian order of the platform. The possible values are:

- **0** – The sequence contains non-unique weights
- **1** – The sequence contains all unique weights
- **2** – The sequence is the identity sequence, for which strings are compared byte for byte.
- **3** – The sequence is NLSCHAR, used for sorting characters in a TIS620-1 (code page 874) Thai database.
- **4** – The sequence is IDENTITY_16BIT, which implements the "CESU-8 Compatibility Encoding Scheme for UTF–16: 8–bit" algorithm as specified in the Unicode Technical Report #26 available at the Unicode Technical Consortium Web site at http://www.unicode.org
- **X'8001'** – The sequence is UCA400_NO, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.00, with normalization implicitly set to ON.
- **X'8002'** – The sequence is UCA400_LTH, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.00, and sorts all Thai characters as per the Royal Thai Dictonary order.
- **X'8003'** – The sequence is UCA400_LSK, which implements the Unicode Collation Algorithm (UCA) based on the Unicode Standard version 4.00, and sorts all Slovakian characters properly.

If you use this internal type information, you need to consider byte reversal when retrieving information for a database on a different platform.

You can specify the collating sequence at database creation time.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

country/region - Database territory code

Configuration Type Database

Parameter Type Informational

This parameter shows the territory code used to create the database.

Related reference:

- "territory - Database territory " on page 472

- "GET DATABASE CONFIGURATION command" in *Command Reference*

database_level - Database release level

Configuration Type Database

Parameter Type Informational

This parameter indicates the release level of the database manager which can use the database. In the case of an incomplete or failed migration, this parameter will reflect the release level of the unmigrated database and might differ from the *release* parameter (the release level of the database configuration file). Otherwise the value of *database_level* will be identical to value of the *release* parameter.

Related reference:

- "release - Configuration file release level " on page 472
- "GET DATABASE CONFIGURATION command" in *Command Reference*

pagesize - Database default page size

Configuration Type Database

Parameter Type Informational

This parameter contains the value that was used as the default page size when the database was created. Possible values are: 4 096, 8 192, 16 384 and 32 768. When a buffer pool or table space is created in that database, the same default page size applies.

Related reference:

- "CREATE DATABASE command" in *Command Reference*

release - Configuration file release level

Configuration Type Database manager, Database

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Informational

This parameter specifies the release level of the configuration file.

Related reference:

- "database_level - Database release level " on page 472
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

territory - Database territory

Configuration Type Database

Parameter Type Informational

This parameter shows the territory used to create the database. *territory* is used by the database manager when processing data that is territory-sensitive.

Related reference:

- "country/region - Database territory code " on page 471
- "GET DATABASE CONFIGURATION command" in *Command Reference*

Status

The following parameters provide information about the state of the database:

- "backup_pending - Backup pending indicator "
- "database_consistent - Database is consistent "
- "log_retain_status - Log retain status indicator "
- "multipage_alloc - Multipage file allocation enabled " on page 474
- "restore_pending - Restore pending " on page 474
- "rollfwd_pending - Roll forward pending indicator " on page 474
- "user_exit_status - User exit status indicator " on page 475

backup_pending - Backup pending indicator

Configuration Type Database

Parameter Type Informational

If set on, this parameter indicates that you must do a full backup of the database before accessing it. This parameter is only on if the database configuration is changed so that the database moves from being nonrecoverable to recoverable (that is, initially both the *logretain* and *userexit* parameters were set to NO, then either one or both of these parameters is set to YES, and the update to the database configuration is accepted).

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

database_consistent - Database is consistent

Configuration Type Database

Parameter Type Informational

This parameter indicates whether the database is in a consistent state.

YES indicates that all transactions have been committed or rolled back so that the data is consistent. If the system "crashes" while the database is consistent, you do not need to take any special action to make the database usable.

NO indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system "crashes" while the database is not consistent, you will need to restart the database using the RESTART DATABASE command to make the database usable.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

log_retain_status - Log retain status indicator

Configuration Type Database

Parameter Type Informational

If set, this parameter indicates that log files are being retained for use in roll-forward recovery.

This parameter is set when the *logretain* parameter setting is equal to Recovery.

Related reference:

- "logretain - Log retain enable " on page 449
- "GET DATABASE CONFIGURATION command" in *Command Reference*

multipage_alloc - Multipage file allocation enabled

Configuration Type Database

Parameter Type Informational

Multipage file allocation is used to improve insert performance. It applies to SMS table spaces only. If enabled, all SMS table spaces are affected: there is no selection possible for individual SMS table spaces.

The default for the parameter is Yes: multipage file allocation is enabled.

Following database creation, this parameter cannot be set to No. Multipage file allocation cannot be disabled once it has been enabled. If multipage file allocation is not desired, the DB2_NO_MPFA_FOR_NEW_DB DB2 registry variable must be set appropriately before the database is created. The **db2empfa** tool can be used to enable multipage file allocation for a database that currently has it disabled.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

restore_pending - Restore pending

Configuration Type Database

Parameter Type Informational

This parameter states whether a RESTORE PENDING status exists in the database.

Related reference:

- "userexit - User exit enable " on page 452
- "GET DATABASE CONFIGURATION command" in *Command Reference*

rollfwd_pending - Roll forward pending indicator

Configuration Type Database

Parameter Type Informational

This parameter can indicate one of the following states:

- **DATABASE**, meaning that a roll-forward recovery procedure is required for this database
- **TABLESPACE**, meaning that one or more table spaces need to be rolled forward
- **NO**, meaning that the database is usable and no roll-forward recovery is required.

The recovery (using ROLLFORWARD DATABASE) must complete before you can access the database or table space.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

user_exit_status - User exit status indicator

Configuration Type	Database
Parameter Type	Informational

If set to On, this indicates that the database manager is enabled for roll-forward recovery and that the user exit program will be used to archive and retrieve log files when called by the database manager.

Related reference:

- "userexit - User exit enable " on page 452
- "GET DATABASE CONFIGURATION command" in *Command Reference*

Compiler settings

The following parameters provide information to influence the compiler:

- "dft_degree - Default degree "
- "dft_mttb_types - Default maintained table types for optimization " on page 476
- "dft_queryopt - Default query optimization class " on page 476
- "dft_refresh_age - Default refresh age " on page 477
- "dft_sqlmathwarn - Continue upon arithmetic exceptions " on page 477
- "num_freqvalues - Number of frequent values retained " on page 479
- "num_quantiles - Number of quantiles for columns " on page 480

dft_degree - Default degree

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	1 [-1, 1 – 32 767]

This parameter specifies the default value for the CURRENT DEGREE special register and the DEGREE bind option.

The default value is 1.

A value of 1 means no intra-partition parallelism. A value of -1 means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query.

The degree of intra-partition parallelism for an SQL statement is specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option. The maximum runtime degree of intra-partition parallelism for an active application is specified using the SET RUNTIME DEGREE command. The Maximum Query Degree of Parallelism (*max_querydegree*) configuration parameter specifies the maximum query degree of intra-partition parallelism for all SQL queries.

The actual runtime degree used is the lowest of:

- *max_querydegree* configuration parameter
- application runtime degree
- SQL statement compilation degree

Related reference:

- "max_querydegree - Maximum query degree of parallelism " on page 492
- "Restrictions on native XML data store" in *XML Guide*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

dft_mttb_types - Default maintained table types for optimization

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	SYSTEM [ALL, NONE, FEDERATED_TOOL, SYSTEM, USER, or a list of values]

This parameter specifies the default value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register. The value of this register determines what types of refresh deferred materialized query tables will be used during query optimization.

You can specify a list of values separated by commas; for example, 'USER,FEDERATED_TOOL'. ALL or NONE cannot be listed with other values, and you cannot specify the same value more than once.

Related reference:

- "CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register" in *SQL Reference, Volume 1*

dft_queryopt - Default query optimization class

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	5 [0 — 9]
Unit of Measure	Query Optimization Class (see below)

The query optimization class is used to direct the optimizer to use different degrees of optimization when compiling SQL and XQuery queries. This parameter provides additional flexibility by setting the default query optimization class used when neither the SET CURRENT QUERY OPTIMIZATION statement nor the QUERYOPT option on the bind command are used.

The query optimization classes currently defined are:

- 0 - minimal query optimization.
- 1 - roughly comparable to DB2 Version 1.
- 2 - slight optimization.
- 3 - moderate query optimization.

- 5 - significant query optimization with heuristics to limit the effort expended on selecting an access plan. This is the default.
- 7 - significant query optimization.
- 9 - maximal query optimization

Related reference:
- "SET CURRENT QUERY OPTIMIZATION statement" in *SQL Reference, Volume 2*
- "BIND command" in *Command Reference*
- "LIST DRDA INDOUBT TRANSACTIONS command" in *Command Reference*
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

dft_refresh_age - Default refresh age

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	0 [0, 99999999999999 (ANY)]

This parameter has the default value used for the REFRESH AGE if the CURRENT REFRESH AGE special register is not specified. This parameter specifies a time stamp duration value with a data type of DECIMAL(20,6). This time duration represents the maximum duration since a REFRESH TABLE statement has been processed on a specific REFRESH DEFERRED materialized query table during which that summary table can be used to optimize the processing of a query. If the CURRENT REFRESH AGE has a value of 99999999999999 (ANY), and the QUERY OPTIMIZATION class has a value of two, or five or more, REFRESH DEFERRED materialized query tables are considered to optimize the processing of a dynamic query.

Related reference:
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "Restrictions on native XML data store" in *XML Guide*

dft_sqlmathwarn - Continue upon arithmetic exceptions

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	No [No, Yes]

This parameter sets the default value that determines the handling of arithmetic errors and retrieval conversion errors as errors or warnings during SQL statement compilation. For static SQL statements, the value of this parameter is associated with the package at bind time. For dynamic SQL DML statements, the value of this parameter is used when the statement is prepared.

Attention: If you change the *dft_sqlmathwarn* value for a database, the behavior of check constraints, triggers, and views that include arithmetic expressions might change. This might, in turn, have an impact on the data integrity of the database. You should only change the setting of *dft_sqlmathwarn* for a database after carefully

evaluating how the new arithmetic exception handling behavior might impact check constraints, triggers, and views. Once changed, subsequent changes require the same careful evaluation.

As an example, consider the following check constraint, which includes a division arithmetic operation:

```
A/B > 0
```

When *dft_sqlmathwarn* is "No" and an INSERT with B=0 is attempted, the division by zero is processed as an arithmetic error. The insert operation fails because DB2 cannot check the constraint. If *dft_sqlmathwarn* is changed to "Yes", the division by zero is processed as an arithmetic warning with a NULL result. The NULL result causes the predicate to evaluate to UNKNOWN and the insert operation succeeds. If *dft_sqlmathwarn* is changed back to "No", an attempt to insert the same row will fail, because the division by zero error prevents DB2 from evaluating the constraint. The row inserted with B=0 when *dft_sqlmathwarn* was "Yes" remains in the table and can be selected. Updates to the row that cause the constraint to be evaluated will fail, while updates to the row that do not require constraint re-evaluation will succeed.

Before changing *dft_sqlmathwarn* from "No" to "Yes", you should consider rewriting the constraint to explicitly handle nulls from arithmetic expressions. For example:

```
( A/B > 0 ) AND ( CASE
                    WHEN A IS NULL THEN 1
                    WHEN B IS NULL THEN 1
                    WHEN A/B IS NULL THEN 0
                    ELSE 1
                    END
                  = 1 )
```

can be used if both A and B are nullable. And, if A or B is not-nullable, the corresponding IS NULL WHEN-clause can be removed.

Before changing *dft_sqlmathwarn* from "Yes" to "No", you should first check for data that might become inconsistent by using, for example, predicates such as the following:

```
WHERE A IS NOT NULL AND B IS NOT NULL AND A/B IS NULL
```

When inconsistent rows are isolated, you should take appropriate action to correct the inconsistency before changing *dft_sqlmathwarn*. You can also manually re-check constraints with arithmetic expressions after the change. To do this, first place the affected tables in a check pending state (with the OFF clause of the SET CONSTRAINTS statement), then request that the tables be checked (with the IMMEDIATE CHECKED clause of the SET CONSTRAINTS statement). Inconsistent data will be indicated by an arithmetic error, which prevents the constraint from being evaluated.

Recommendation: Use the default setting of no, unless you specifically require queries to be processed that include arithmetic exceptions. Then specify the value of yes. This situation can occur if you are processing SQL statements that, on other database managers, provide results regardless of the arithmetic exceptions that occur.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*

- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

num_freqvalues - Number of frequent values retained

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [0 — 32 767]
Unit of Measure	Counter

This parameter allows you to specify the number of "most frequent values" that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command. Increasing the value of this parameter increases the amount of statistics heap (*stat_heap_sz*) used when collecting statistics.

The "most frequent value" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the query optimizer but requires additional catalog space. When 0 is specified, no frequent-value statistics are retained, even if you request that distribution statistics be collected.

You can also specify the number of frequent values retained as part of the RUNSTATS command at the table or the column level. If none is specified, the *num_freqvalues* configuration parameter value is used.

Updating this parameter can help the optimizer obtain better selectivity estimates for some predicates (=, <, >, IS NULL, IS NOT NULL) over data that is non-uniformly distributed. More accurate selectivity calculations might result in the choice of more efficient access plans.

After changing the value of this parameter, you need to:

- Run the RUNSTATS command after all users have disconnected from the database and you have reconnected to the database
- Rebind any packages containing static SQL or XQuery statements.

The RUNSTATS command allows for the specification of the number of frequent values retained, by using the NUM_REQVALUES option. Changing the number of frequent values retained through the RUNSTATS command is easier than making the change using the *num_freqvalues* database configuration parameter.

When using RUNSTATS, you have the ability to limit the number of frequent values collected at both the table level and the column level. This allows you to optimize on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

Recommendation: In order to update this parameter you should determine the degree of non-uniformity in the most important columns (in the most important tables) that typically have selection predicates. This can be done using an SQL SELECT statement that provides an ordered ranking of the number of occurrences of each value in a column. You should not consider uniformly distributed, unique, long, or LOB columns. A reasonable practical value for this parameter lies in the range of 10 to 100.

Note that the process of collecting frequent value statistics requires significant CPU and memory (*stat_heap_sz*) resources.

Related reference:

- "num_quantiles - Number of quantiles for columns " on page 480
- "stat_heap_sz - Statistics heap size " on page 402
- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

num_quantiles - Number of quantiles for columns

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	20 [0 – 32 767]
Unit of Measure	Counter

This parameter controls the number of quantiles that will be collected when the WITH DISTRIBUTION option is specified on the RUNSTATS command. Increasing the value of this parameter increases the amount of statistics heap (*stat_heap_sz*) used when collecting statistics.

The "quantile" statistics help the optimizer understand the distribution of data values within a column. A higher value results in more information being available to the query optimizer but requires additional catalog space. When 0 or 1 is specified, no quantile statistics are retained, even if you request that distribution statistics be collected.

You can also specify the number of quantiles collected as part of the RUNSTATS command at the table or the column level. If none is specified, the *num_quantiles* configuration parameter value is used.

Updating this parameter can help obtain better selectivity estimates for range predicates over data that is non-uniformly distributed. Among other optimizer decisions, this information has a strong influence on whether an index scan or a table scan will be chosen. (It is more efficient to use a table scan to access a range of values that occur frequently and it is more efficient to use an index scan for a range of values that occur infrequently.)

After changing the value of this parameter, you need to:

- Run the RUNSTATS command after all users have disconnected from the database and you have reconnected to the database
- Rebind any packages containing static SQL or XQuery statements.

The RUNSTATS command allows for the specification of the number of quantiles that will be collected, by using the NUM_QUANTILES option. Changing the number of quantiles that will be collected through the RUNSTATS command is easier than making the change using the *num_quantiles* database configuration parameter.

When using RUNSTATS, you have the ability to limit the number of quantiles collected at both the table level and the column level. This allows you to optimize

on space occupied in the catalogs by reducing the distribution statistics for columns where they could not be exploited and yet still using the information for critical columns.

Recommendation: This default value for this parameter guarantees a maximum estimation error of approximately 2.5% for any single-sided range predicate (>, >=, <, or <=), and a maximum error of 5% for any BETWEEN predicate. A simple way to approximate the number of quantiles is:

- Determine the maximum error that is tolerable in estimating the number of rows of any range query, as a percentage, P
- The number of quantiles should be approximately 100/P if most of your predicates are BETWEEN predicates, and 50/P if most of your predicates are other types of range predicates (<, <=, >, or >=).

For example, 25 quantiles should result in a maximum estimate error of 4% for BETWEEN predicates and of 2% for ″>″ predicates. A reasonable practical value for this parameter lies in the range of 10 to 50.

Related reference:

- “num_freqvalues - Number of frequent values retained ” on page 479
- “stat_heap_sz - Statistics heap size ” on page 402
- “GET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION command” in *Command Reference*

Automated maintenance

The following parameter allows you to control the automatic maintenance activities of several DB2 utilities:

- “auto_maint - Automatic maintenance ”

auto_maint - Automatic maintenance

Configuration Type Database

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] ON [ON; OFF]

This parameter is the parent of all the other automatic maintenance database configuration parameters (*auto_db_backup*, *auto_tbl_maint*, *auto_runstats*, *auto_stats_prof*, *auto_prof_upd*, and *auto_reorg*). When this parameter is disabled, all of its children parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its children parameters take effect. In this way, automatic maintenance can be enabled or disabled globally.

By default, this parameter is set to ON.

You can enable or disable individual automatic maintenance features independently by setting the following parameters:

auto_db_backup
This automated maintenance parameter enables or disables automatic backup operations for a database. A backup policy (a defined set of rules or guidelines) can be used to specify the automated behavior. The objective of the backup policy is to ensure that the database is being backed up regularly. The backup policy for a database is created automatically when the DB2 Health Monitor first runs. By default, this parameter is set to OFF. To be enabled, this parameter must be set to ON, and its parent parameter must also be enabled.

auto_tbl_maint
This parameter is the parent of all table maintenance parameters (*auto_runstats*, *auto_stats_prof*, *auto_prof_upd*, and *auto_reorg*). When this parameter is disabled, all of its children parameters are also disabled, but their settings, as recorded in the database configuration file, do not change. When this parent parameter is enabled, recorded values for its children parameters take effect. In this way, table maintenance can be enabled or disabled globally.

By default, this parameter is set to ON.

auto_runstats
This automated table maintenance parameter enables or disables automatic table runstats operations for a database. A runstats policy (a defined set of rules or guidelines) can be used to specify the automated behavior. Statistics collected by the runstats utility are used by the optimizer to determine the most efficient plan for accessing the physical data. To be enabled, this parameter must be set to On, and its parent parameters must also be enabled.

By default, this parameter is set to ON.

auto_stats_prof
When enabled, this automated table maintenance parameter turns on statistical profile generation, designed to improve applications whose workloads include complex queries, many predicates, joins, and grouping operations over several tables. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

auto_prof_upd
When enabled, this automated table maintenance parameter (a child of *auto_stats_prof*) specifies that the runstats profile is to be updated with recommendations. When this parameter is disabled, recommendations are stored in the opt_feedback_ranking table, which you can inspect

when manually updating the runstats profile. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

auto_reorg

This automated table maintenance parameter enables or disables automatic table and index reorganization for a database. A reorganization policy (a defined set of rules or guidelines) can be used to specify the automated behavior. To be enabled, this parameter must be set to ON, and its parent parameters must also be enabled.

By default, this parameter is set to OFF.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

Communications

The following groups of parameters provide information about using DB2 in a client/server environment:

- "Communication protocol setup"
- "DB2 Discovery" on page 485

Communication protocol setup

You can use the following parameters to configure your database clients and database servers:

- "nname - NetBIOS workstation name "
- "svcename - TCP/IP service name " on page 484

nname - NetBIOS workstation name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter allows you to assign a unique name to the database instance on a workstation in the NetBIOS LAN environment. This *nname* is the basis for the actual NetBIOS names that will be registered with NetBIOS for a workstation.

Since the NetBIOS protocol establishes connections using these NetBIOS names, the *nname* parameter must be set for both the client and server.

Client applications must know the *nname* of the server that contains the database to be accessed. The server's *nname* must be cataloged in the client's node directory as the "server-nname" parameter using the CATALOG NETBIOS NODE command.

If *nname* at the server node changes to a new name, all clients accessing databases on that server must catalog this new name for the server.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

svcename - TCP/IP service name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter contains the name of the TCP/IP port which a database server will use to await communications from remote client nodes. This name must be the first of two consecutive ports reserved for use by the database manager; the second port is used to handle interrupt requests from down-level clients.

In order to accept connection requests from a database client using TCP/IP, the database server must be listening on a port designated to that server. The system administrator for the database server must reserve a port (number *n*) and define its associated TCP/IP service name in the services file at the server. If the database server needs to support requests from down-level clients, a second port (number *n+1*, for interrupt requests) needs to be defined in the services file at the server.

The database server port (number *n*) and its TCP/IP service name need to be defined in the services file on the database client. Down-level clients also require the interrupt port (number *n+1*) to be defined in the client's services file.

On UNIX-based systems, the services file is located in: `/etc/services`

The *svcename* parameter should be set to the service name associated with the main connection port so that when the database server is started, it can determine on which port to listen for incoming connection requests. If you are supporting or using a down-level client, the service name for the interrupt port is not saved in the configuration file. The interrupt port number can be derived based on the main connection port number (*interrupt port number = main connection port* + 1).

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

DB2 Discovery

You can use the following parameters to establish DB2 Discovery:

- "discover - Discovery mode "
- "discover_db - Discover database "
- "discover_inst - Discover server instance " on page 486

discover - Discovery mode

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] SEARCH [DISABLE, KNOWN, SEARCH]

From a client perspective, one of the following will occur:

- If *discover* = SEARCH, the client can issue search discovery requests to find DB2 server systems on the network. Search discovery provides a superset of the functionality provided by KNOWN discovery. If *discover* = SEARCH, both search and known discovery requests can be issued by the client.
- If *discover* = KNOWN, only known discovery requests can be issued from the client. By specifying some connection information for the administration server on a particular system, all the instance and database information on the DB2 system is returned to the client.
- If *discover* = DISABLE, discovery is disabled at the client.

The default discovery mode is SEARCH.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

discover_db - Discover database

Configuration Type Database

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] Enable [Disable, Enable]

This parameter is used to prevent information about a database from being returned to a client when a discovery request is received at the server.

The default for this parameter is that discovery is enabled for this database.

By changing this parameter value to "Disable", it is possible to hide databases with sensitive data from the discovery process. This can be done in addition to other database security controls on the database.

Related reference:

- "GET DATABASE CONFIGURATION command" in *Command Reference*
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*

discover_inst - Discover server instance

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable online

Propagation Class Immediate

Default [Range] ENABLE [ENABLE, DISABLE]

This parameter specifies whether this instance can be detected by DB2 discovery. The default, enable, specifies that the instance can be detected, while disable prevents the instance from being discovered.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Partitioned database environment

The following groups of parameters provide information about parallel operations and partitioned database environments:

- "Communications"
- "Parallel processing" on page 491

Communications

The following parameters provide information about communications in the partitioned database environment:

- "conn_elapse - Connection elapse time "
- "fcm_num_buffers - Number of FCM buffers "
- "fcm_num_channels - Number of FCM channels " on page 488
- "max_connretries - Node connection retries " on page 489
- "max_time_diff - Maximum time difference among nodes " on page 490
- "start_stop_time - Start and stop timeout " on page 490

conn_elapse - Connection elapse time

Configuration Type Database manager

Applies To Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] 10 [0–100]

Unit of Measure Seconds

This parameter specifies the number of seconds within which a TCP/IP connection is to be established between two database partition servers. If the attempt completes within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the *max_connretries* parameter and always times out, an error is issued.

Related reference:

- "max_connretries - Node connection retries " on page 489
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

fcm_num_buffers - Number of FCM buffers

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

32-bit platforms
Automatic [Automatic, 128 — *65 300*]

64-bit platforms
Automatic [Automatic, 128 — *524 288*]

- Database server with local and remote clients: the default is 1 024
- Database server with local clients: the default is 512
- Partitioned database server with local and remote clients: the default is 4 096

On single-partition database systems, this parameter is not used if the *intra_parallel* parameter is not active.

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within database servers.

If you have multiple logical nodes on the same machine, you might find it necessary to increase the value of this parameter. You might also find it necessary to increase the value of this parameter if you run out of message buffers because of the number of users on the system, the number of database partition servers on the system, or the complexity of the applications.

If you are using multiple logical nodes, one pool of *fcm_num_buffers* buffers is shared by all the multiple logical nodes on the same machine. The size of the buffer pool will be determined by multiplying the fcm_num_buffersvalue times the number of logical nodes on that physical machine. Re-examine the value you are using; consider how many FCM buffers in total will be allocated on the machine (or machines) where the multiple logical nodes reside.

Related concepts:

- "Fast communications manager (FCM) communications" in *Administration Guide: Implementation*

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

fcm_num_channels - Number of FCM channels

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

UNIX 32-bit platforms
Automatic, with starting values of 256, 512, 2 048 [128 — 120 000]

UNIX 64-bit platforms
Automatic, with starting values of 256, 512, 2 048 [128 — 524 288]

Windows 32-bit
Automatic, with a starting value 10 000 [128 — 120 000]

Windows 64-bit
Automatic, with starting values of 256, 512, 2 048 [128 — 524 288]

- For database server with local and remote clients, the starting value is 512.
- For database server with local clients, the starting value is 256.
- For partitioned database server with local and remote clients, the starting value is 2 048.

On non-partitioned database systems, the *intra_parallel* parameter must be active before *fcm_num_channels* can be used.

An FCM channel represents a logical communication end point between EDUs running in the DB2 engine. Both control flows (request and reply) and data flows (table queue data) rely on channels to transfer data between partitions. This parameter specifies the number of FCM channels for each database partition.

Related reference:

- "ch_free - Channels Currently Free monitor element" in *System Monitor Guide and Reference*
- "ch_free_bottom - Minimum Channels Free monitor element" in *System Monitor Guide and Reference*

max_connretries - Node connection retries

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	5 [0–100]

If the attempt to establish communication between two database partition servers fails (for example, the value specified by the *conn_elapse* parameter is reached), *max_connretries* specifies the number of connection retries that can be made to a database partition server. If the value specified for this parameter is exceeded, an error is returned.

Related reference:

- "conn_elapse - Connection elapse time " on page 487

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

max_time_diff - Maximum time difference among nodes

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	60 [1–1 440]
Unit of Measure	Minutes

Each database partition server has its own system clock. This parameter specifies the maximum time difference, in minutes, that is permitted among the database partition servers listed in the node configuration file.

If two or more database partition servers are associated with a transaction, and their clocks are not synchronized to within the time specified by this parameter, the transaction is rejected and an SQLCODE is returned. (The transaction is rejected only if data modification is associated with it.)

DB2 uses *Coordinated Universal Time* (UTC), so different time zones are not a consideration when you set this parameter. The Coordinated Universal Time is the same as Greenwich Mean Time.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

start_stop_time - Start and stop timeout

Configuration Type	Database manager
Applies To	Database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [1 — 1 440]
Unit of Measure	Minutes

This parameter specifies the time, in minutes, within which all database partition servers must respond to a DB2START or a DB2STOP command. It is also used as the timeout value during an ADD DBPARTITIONNUM operation.

Database partition servers that do not respond to a DB2START command within the specified time send a message to the `db2start` error log in the `log` subdirectory

of the sqllib subdirectory of the home directory for the instance. You should issue a DB2STOP on these nodes before restarting them.

Database partition servers that do not respond to a DB2STOP command within the specified time send a message to the db2stop error log in the log subdirectory of the sqllib subdirectory of the home directory for the instance. You can either issue db2stop for each database partition server that does not respond, or for all of them. (Those that are already stopped will return stating that they are stopped.)

If a db2start or db2stop operation in a multi-partition database is not completed within the value specified by the *start_stop_timeout* database manager configuration parameter, the database partitions that have timed out will be killed internally. Environments with many database partitions with a low value for *start_stop_timeout* might experience this behavior. To resolve this behavior, increase the value of *start_stop_timeout*.

When adding a new database partition using one of the DB2START, START DATABASE MANAGER, or ADD DBPARTITIONNUM commands, the add database partition operation must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for each database. If automatic storage is enabled, the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the operation might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. These factors should be considered when determining the value of the *start_stop_time* parameter.

Related reference:

- "ADD DBPARTITIONNUM command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Parallel processing

The following parameters provide information about parallel processing:

- "intra_parallel - Enable intra-partition parallelism "
- "max_querydegree - Maximum query degree of parallelism " on page 492

intra_parallel - Enable intra-partition parallelism

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] NO (0) [SYSTEM (-1), NO (0), YES (1)]

A value of -1 causes the parameter value to be set to “YES” or “NO” based on the hardware on which the database manager is running.

This parameter specifies whether the database manager can use intra-partition parallelism.

Some of the operations that can take advantage of parallel performance improvements when this parameter is ″YES″ include database queries and index creation.

Note: If you change this parameter value, packages might be rebound to the database, and some performance degradation might occur.

Related reference:

- “max_querydegree - Maximum query degree of parallelism ” on page 492
- “GET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE MANAGER CONFIGURATION command” in *Command Reference*

max_querydegree - Maximum query degree of parallelism

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	-1 (ANY) [ANY, 1 — 32 767] (ANY means system determined)

This parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a database partition when the statement is executed. The *intra_parallel* configuration parameter must be set to “YES” to enable the database partition to use intra-partition parallelism.

The default value for this configuration parameter is -1. This value means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.

Note: The degree of parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option.

The maximum query degree of parallelism for an active application can be modified using the SET RUNTIME DEGREE command. The actual runtime degree used is the lower of:

- *max_querydegree* configuration parameter
- Application runtime degree
- SQL statement compilation degree

An exception regarding the determination of the actual query degree of parallelism occurs when creating an index. In this case, if *intra_parallel* is "YES" and the table is large enough to benefit from the use of multiple processors, then creating an index uses the number of online processors (to a maximum of 6) plus one. There is no effect from the other parameter, bind option, or special register mentioned above.

Related reference:

- "dft_degree - Default degree " on page 475
- "intra_parallel - Enable intra-partition parallelism " on page 491
- "Restrictions on native XML data store" in *XML Guide*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Instance management

A number of parameters can help you manage your database manager instances. These are grouped into the following categories:

- "Diagnostic"
- "Database system monitor parameters" on page 497
- "System management" on page 498
- "Instance administration" on page 506

Diagnostic

The following parameters allow you to control diagnostic information available from the database manager:

- "diaglevel - Diagnostic error capture level "
- "diagpath - Diagnostic data directory path " on page 494
- "health_mon - Health monitoring " on page 495
- "notifylevel - Notify level " on page 496

diaglevel - Diagnostic error capture level

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] 3 [0 — 4]

This parameter specifies the type of diagnostic errors that will be recorded in the db2diag.log file. Valid values are:

0 – No diagnostic data captured

1 – Severe errors only

2 – All errors

3 – All errors and warnings

4 – All errors, warnings and informational messages

The *diagpath* configuration parameter is used to specify the directory that will contain the error file, alert log file, and any dump files that might be generated, based on the value of the *diaglevel* parameter.

Recommendation: You might wish to increase the value of this parameter to gather additional problem determination data to help resolve a problem.

Related reference:

- "diagpath - Diagnostic data directory path " on page 494
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

diagpath - Diagnostic data directory path

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range] Null [any valid path name]

This parameter allows you to specify the fully qualified path for DB2 diagnostic information. This directory could possibly contain dump files, trap files, an error log, a notification file, and an alert log file, depending on your platform.

If this parameter is null, the diagnostic information will be written to files in one of the following directories or folders:

- For supported Windows environments:

  - If the DB2INSTPROF environment variable or keyword is **not** set, information will be written to `x:\SQLLIB\DB2INSTANCE`, where `x:\SQLLIB` is the drive reference and directory specified in the DB2PATH registry variable or environment variable, and `DB2INSTANCE` is the name of the instance owner.

    Note: The directory does not have to be named SQLLIB.
  - If the DB2INSTPROF environment variable or keyword is set, information will be written to `x:\DB2INSTPROF\DB2INSTANCE`, where DB2INSTPROF is the name of the instance profile directory and DB2INSTANCE is the name of the instance.
- For UNIX-based environments: `INSTHOME/sqllib/db2dump`, where `INSTHOME` is the home directory of the instance.

Recommendation: Use the default or have a centralized location for the diagpath of multiple instances.

In a partitioned database environment, the path you specify must reside on a shared file system.

Related reference:

- "diaglevel - Diagnostic error capture level " on page 493
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

health_mon - Health monitoring

Configuration Type	Database manager
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	On [On; Off]
Related Parameters	

This parameter allows you to specify whether you want to monitor an instance, its associated databases, and database objects according to various health indicators. If *health_mon* is turned on (the default), an agent will collect information about the health of the objects you have selected. If an object is considered to be in an unhealthy position, based on thresholds that you have set, notifications can be sent, and actions can be taken automatically. If *health_mon* is turned off, the health of objects will not be monitored.

You can use the Health Center or the CLP to select the instance and database objects that you want to monitor. You can also specify where notifications should be sent, and what actions should be taken, based on the data collected by the health monitor.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

notifylevel - Notify level

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	3 [0 — 4]

This parameter specifies the type of administration notification messages that are written to the administration notification log. On UNIX platforms, the administration notification log is a text file called *instance*`.nfy`. On Windows, all administration notification messages are written to the Event Log. The errors can be written by DB2, the Health Monitor, the Capture and Apply programs, and user applications.

Valid values for this parameter are:

0 — No administration notification messages captured. (This setting is not recommended.)

1 — Fatal or unrecoverable errors. Only fatal and unrecoverable errors are logged. To recover from some of these conditions, you might need assistance from DB2 service.

2 — Immediate action required. Conditions are logged that require immediate attention from the system administrator or the database administrator. If the condition is not resolved, it could lead to a fatal error. Notification of very significant, non-error activities (for example, recovery) might also be logged at this level. This level will capture Health Monitor alarms.

3 — Important information, no immediate action required. Conditions are logged that are non-threatening and do not require immediate action but might indicate a non-optimal system. This level will capture Health Monitor alarms, Health Monitor warnings, and Health Monitor attentions.

4 — Informational messages.

The administration notification log includes messages having values up to and including the value of *notifylevel*. For example, setting *notifylevel* to 3 will cause the administration notification log to include messages applicable to levels 1, 2, and 3.

For a user application to be able to write to the notification file or Windows Event Log, it must call the db2AdminMsgWrite API.

Recommendation: You might wish to increase the value of this parameter to gather additional problem determination data to help resolve a problem. Note that

you must set *notifylevel* to a value of 2 or higher for the Health Monitor to send any notifications to the contacts defined in its configuration.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Database system monitor parameters

The following parameter allows you to control various aspects of the database system monitor:

- "dft_monswitches - Default database system monitor switches "

dft_monswitches - Default database system monitor switches

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default All switches turned off, except dft_mon_timestamp, which is turned on by default

This parameter is unique in that it allows you to set a number of switches which are each internally represented by a bit of the parameter. You can update each of these switches independently by setting the following parameters:

dft_mon_uow Default value of the snapshot monitor's unit of work (UOW) switch

dft_mon_stmt Default value of the snapshot monitor's statement switch

dft_mon_table Default value of the snapshot monitor's table switch

dft_mon_bufpool Default value of the snapshot monitor's buffer pool switch

dft_mon_lock Default value of the snapshot monitor's lock switch

dft_mon_sort Default value of the snapshot monitor's sort switch

dft_mon_timestamp Default value of the snapshot monitor's timestamp switch

Recommendation: Any switch (except dft_mon_timestamp) that is turned ON instructs the database manager to collect monitor data related to that switch. Collecting additional monitor data increases database manager overhead which can impact system performance. Turning the dft_mon_timestamp switch OFF becomes

important as CPU utilization approaches 100%. When this occurs, the CPU time required for issuing timestamps increases dramatically. Furthermore, if the timestamp switch is turned OFF, the overall cost of other data under monitor switch control is greatly reduced.

All monitoring applications inherit these default switch settings when the application issues its first monitoring request (for example, setting a switch, activating the event monitor, taking a snapshot). You should turn on a switch in the configuration file only if you want to collect data starting from the moment the database manager is started. (Otherwise, each monitoring application can set its own switches and the data it collects becomes relative to the time its switches are set.)

Related reference:

- "GET MONITOR SWITCHES command" in *Command Reference*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

System management

The following parameters relate to system management:

- "comm_bandwidth - Communications bandwidth "
- "cpuspeed - CPU speed " on page 499
- "dft_account_str - Default charge-back account " on page 500
- "federated - Federated database system support " on page 501
- "jdk_path - Software Developer's Kit for Java installation path " on page 501
- "nodetype - Machine node type " on page 502
- "numdb - Maximum number of concurrently active databases including host and iSeries databases " on page 502
- "tp_mon_name - Transaction processor monitor name " on page 503
- "util_impact_lim - Instance impact policy " on page 505

comm_bandwidth - Communications bandwidth

Configuration Type	Database manager
Applies to	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Statement boundary
Default [Range]	-1 [.1 – 100 000] A value of -1 causes the parameter value to be reset to the default. The default value is calculated based on whether a high speed switch is being used.
Unit of Measure	Megabytes per second

The value calculated for the communications bandwidth, in megabytes per second, is used by the query optimizer to estimate the cost of performing certain operations between the database partition servers of a partitioned database system. The optimizer does not model the cost of communications between a client and a server, so this parameter should reflect only the nominal bandwidth between the database partition servers, if any.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The communications bandwidth is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

cpuspeed - CPU speed

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable online
Propagation Class	Statement boundary
Default [Range]	-1 [$1x10^{-10}$ — 1] A value of -1 will cause the parameter value to be reset based on the running of the measurement program.
Unit of Measure	Seconds

The CPU speed, in milliseconds per instruction, is used by the query optimizer to estimate the cost of performing certain operations. The value of this parameter is set automatically when you install the database manager based on the output from a program designed to measure CPU speed. This program is executed if benchmark results are not available for any of the following reasons:

- The platform does not have support for the `db2spec.dat` file
- The `db2spec.dat` file is **not** found
- The data for the IBM RISC System/6000® model 530H is not found in the file
- The data for your machine is not found in the file.

You can explicitly set this value to model a production environment on your test system or to assess the impact of upgrading hardware. By setting it to -1, *cpuspeed* will be re-computed.

Recommendation: You should only adjust this parameter if you want to model a different environment.

The CPU speed is used by the optimizer in determining access paths. You should consider rebinding applications (using the REBIND PACKAGE command) after changing this parameter.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

dft_account_str - Default charge-back account

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid string]

With each application connect request, an accounting identifier consisting of a DB2 Connect-generated prefix and the user supplied suffix is sent from the application requester to a DRDA application server. This accounting information provides a mechanism for system administrators to associate resource usage with each user access.

Note: This parameter is only applicable to DB2 Connect.

The suffix is supplied by the application program calling the `sqlesact()` API or the user setting the environment variable DB2ACCOUNT. If a suffix is not supplied by either the API or environment variable, DB2 Connect uses the value of this parameter as the default suffix value. This parameter is particularly useful for down-level database clients (anything prior to version 2) that do not have the capability to forward an accounting string to DB2 Connect.

Recommendation: Set this accounting string using the following:

- Alphabetics (A through Z)
- Numerics (0 through 9)
- Underscore (_).

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

federated - Federated database system support

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] No [Yes; No]

This parameter enables or disables support for applications submitting distributed requests for data managed by data sources (such as the DB2 Family and Oracle).

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

jdk_path - Software Developer's Kit for Java installation path

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [Valid path]

This parameter specifies the directory under which the Software Developer's Kit (SDK) for Java, to be used for running Java stored procedures and user-defined functions, is installed. The CLASSPATH and other environment variables used by the Java interpreter are computed from the value of this parameter.

Because there is no default value for this parameter, you should specify a value when you install the SDK for Java.

Related reference:

- "java_heap_sz - Maximum Java interpreter heap size " on page 410

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

nodetype - Machine node type

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Informational

This parameter provides information about the DB2 products which you have installed on your machine and, as a result, information about the type of database manager configuration. The following are the possible values returned by this parameter and the products associated with that node type:

- **Database server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers.
- **Client** – a database client capable of accessing remote database servers.
- **Database server with local clients** – a DB2 relational database management system, supporting local database clients and capable of accessing other, remote database servers.
- **Partitioned database server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers, and capable of parallelism.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

numdb - Maximum number of concurrently active databases including host and iSeries databases

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range]

UNIX 8 [1 — 256]

Windows Database server with local and remote clients 8 [1 — 256]

Windows Database server with local clients 3 [1 — 256]

Unit of Measure Counter

This parameter specifies the number of local databases that can be concurrently active (that is, have applications connected to them), or the maximum number of different database aliases that can be cataloged on a DB2 Connect server. Each database takes up storage, and an active database uses a new shared memory segment.

Recommendation: It is generally best to set this value to the actual number of databases that are already defined to the database manager, and to add about 10% to this value to allow for growth.

Changing the *numdb* parameter can impact the total amount of memory allocated. As a result, frequent updates to this parameter are not recommended. When updating this parameter, you should consider the other configuration parameters that can allocate memory for a database or an application connected to that database.

Related reference:

- "app_ctl_heap_sz - Application control heap size " on page 392
- "applheapsz - Application heap size " on page 397
- "aslheapsz - Application support layer heap size " on page 403
- "dbheap - Database heap " on page 382
- "database_memory - Database shared memory size " on page 380
- "mon_heap_sz - Database system monitor heap size " on page 411
- "locklist - Maximum storage for lock list " on page 383
- "sortheap - Sort heap size " on page 400
- "stmtheap - Statement heap size " on page 402
- "stat_heap_sz - Statistics heap size " on page 402
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

tp_mon_name - Transaction processor monitor name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default No default

Valid Values

- CICS®
- MQ
- ENCINA
- CB
- SF
- TUXEDO
- TOPEND
- blank or some other value (for UNIX and Windows; no other possible values for Solaris or SINIX)

This parameter identifies the name of the transaction processing (TP) monitor product being used.

- If applications are run in a WebSphere® Enterprise Edition CICS environment, this parameter should be set to "CICS"
- If applications are run in a WebSphere Enterprise Edition Encina® environment, this parameter should be set to "ENCINA"
- If applications are run in a WebSphere Enterprise Edition Component Broker environment, this parameter should be set to "CB"
- If applications are run in an IBM MQSeries® environment, this parameter should be set to "MQ"
- If applications are run in a BEA Tuxedo environment, this parameter should be set to "TUXEDO"
- If applications are run in an IBM San Francisco environment, this parameter should be set to "SF".

IBM WebSphere EJB and **Microsoft® Transaction Server** users do not need to configure any value for this parameter.

If none of the above products are being used, this parameter should not be configured but left blank.

In previous versions of IBM DB2 on Windows NT®, this parameter contained the path and name of the DLL which contained the XA Transaction Manager's functions *ax_reg* and *ax_unreg*. This format is still supported. If the value of this parameter does not match any of the above TP Monitor names, it will be assumed that the value is a library name which contains the *ax_reg* and *ax_unreg* functions. This is true for UNIX and Windows NT environments.

TXSeries® CICS and Encina Users: In previous versions of this product on Windows NT it was required to configure this parameter as "libEncServer:C" or "libEncServer:E". While this is still supported, it is no longer required. Configuring the parameter as "CICS" or "ENCINA" is sufficient.

MQSeries Users: In previous versions of this product on Windows NT it was required to configure this parameter as "mqmax". While this is still supported, it is no longer required. Configuring the parameter as "MQ" is sufficient.

Component Broker Users: In previous versions of this product on Windows NT it was required to configure this parameter as "somtrx1i". While this is still supported, it is no longer required. Configuring the parameter as "CB" is sufficient.

San Francisco Users: In previous versions of this product on Windows NT it was required to configure this parameter as "ibmsfDB2". While this is still supported, it is no longer required. Configuring the parameter as "SF" is sufficient.

The maximum length of the string that can be specified for this parameter is 19 characters.

It is also possible to configure this information in IBM DB2 Version 9.1's XA OPEN string. If multiple Transaction Processing Monitors are using a single DB2 instance, then it will be required to use this capability.

Related reference:

- "xa_open string formats" in *Administration Guide: Planning*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

util_impact_lim - Instance impact policy

Configuration Type	Database manager
Applies to	• Database server with local clients • Database server with local and remote clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [1 - 100]
Unit of Measure	Percentage of allowable impact on workload

This parameter allows the database administrator (DBA) to limit the performance degradation of a throttled utility on the workload. The DBA can then run online utilities during critical production periods, and be guaranteed that the performance impact on production work will be within acceptable limits.

For example, a DBA specifying a *util_impact_lim* (impact policy) value of 10 can expect that a throttled backup invocation will not impact the workload by more than 10 percent.

If *util_impact_lim* is 100, no utility invocations will be throttled. In this case, the utilities can have an arbitrary (and undesirable) impact on the workload. If *util_impact_lim* is set to a value that is less than 100, it is possible to invoke utilities in throttled mode. To run in throttled mode, a utility must also be invoked with a non-zero priority.

Recommendation: Most users will benefit from setting *util_impact_lim* to a low value (for example, between 1 and 10).

A throttled utility will usually take longer to complete than an unthrottled utility. If you find that a utility is running for an excessively long time, increase the value of *util_impact_lim*, or disable throttling altogether by setting *util_impact_lim* to 100.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

Instance administration

The following parameters relate to security and administration of your database manager instance:

- "authentication - Authentication type "
- "catalog_noauth - Cataloging allowed without authority " on page 508
- "clnt_krb_plugin - Client Kerberos plug-in " on page 509
- "clnt_pw_plugin - Client userid-password plug-in " on page 509
- "dftdbpath - Default database path " on page 510
- "fed_noauth - Bypass federated authentication " on page 510
- "group_plugin - Group plug-in " on page 511
- "local_gssplugin - GSS API plug-in used for local instance level authorization " on page 512
- "srvcon_auth - Authentication type for incoming connections at the server " on page 512
- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server " on page 513
- "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server " on page 513
- "srv_plugin_mode - Server plug-in mode " on page 514
- "sysadm_group - System administration authority group name " on page 514
- "sysctrl_group - System control authority group name " on page 515
- "sysmaint_group - System maintenance authority group name " on page 516
- "sysmon_group - System monitor authority group name " on page 517
- "trust_allclnts - Trust all clients " on page 518
- "trust_clntauth - Trusted clients authentication " on page 519
- "restrict_access - Database has restricted access configuration parameter" on page 519

authentication - Authentication type

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client

- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DATA_ENCRYPT; DATA_ENCRYPT_CMP KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

This parameter specifies and determines how and where authentication of a user takes place.

If authentication is SERVER, the user ID and password are sent from the client to the server so that authentication can take place on the server. The value SERVER_ENCRYPT provides the same behavior as SERVER, except that any passwords sent over the network are encrypted.

A value of DATA_ENCRYPT means the server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works exactly the same way as SERVER_ENCRYPT.

The following user data are encrypted when using this authentication type:

- SQL statements
- SQL program variable data
- Output data from the server processing an SQL statement and including a description of the data
- Some or all of the answer set data resulting from a query
- Large object (LOB) streaming
- SQLDA descriptors

A value of DATA_ENCRYPT_CMP means the server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with earlier products that do not support DATA_ENCRYPT authentication type. These products are permitted to connect with the SERVER_ENCRYPT authentication type and without encrypting user data. Products supporting the new authentication type must use it. This authentication type is only valid in the server's database manager configuration file and is not valid when used on the CATALOG DATABASE command.

Note: For a Common Criteria compliant configuration, SERVER is the only supported value.

A value of CLIENT indicates that all authentication takes place at the client. No authentication needs to be performed at the server.

A value of KERBEROS means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication. With an authentication type of KRB_SERVER_ENCRYPT at the server and clients that support the Kerberos security system, the effective system authentication type is KERBEROS. If the clients do not support the Kerberos security system, the system authentication type is effectively equivalent to SERVER_ENCRYPT.

A value of GSSPLUGIN means that authentication is performed using an external GSSAPI-based security mechanism. With an authentication type of GSS_SERVER_ENCRYPT at the server and clients that support the GSSPLUGIN security mechanism, the effective system authentication type is GSSPLUGIN (that is, if the clients support one of the server's plug-ins). If the clients do not support the GSSPLUGIN security mechanism, the system authentication type is effectively equivalent to SERVER_ENCRYPT.

Recommendation: Typically, the default value (SERVER) is adequate.

Related concepts:

- "Authentication methods for your server" in *Administration Guide: Implementation*

Related reference:

- "Restrictions on native XML data store" in *XML Guide*
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

catalog_noauth - Cataloging allowed without authority

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

Database server with local and remote clients
NO [NO (0) — YES (1)]

Client; Database server with local clients
YES [NO (0) — YES (1)]

This parameter specifies whether users are able to catalog and uncatalog databases and nodes, or DCS and ODBC directories, without SYSADM authority. The default value (0) for this parameter indicates that SYSADM authority is required. When this parameter is set to 1 (yes), SYSADM authority is not required.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

clnt_krb_plugin - Client Kerberos plug-in

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null or IBMkrb5 [any valid string]

This parameter specifies the name of the default Kerberos plug-in library to be used for client-side authentication and local authorization. By default, the value is null on UNIX-based systems, and `IBMkrb5` on Windows operating systems. This plug-in is used when the client is authenticated using KERBEROS authentication.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

clnt_pw_plugin - Client userid-password plug-in

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the userid-password plug-in library to be used for client-side authentication and local authorization. By default, the value is null and the DB2-supplied userid-password plug-in library is used. The plug-in is used when the client is authenticated using CLIENT, SERVER, or SERVER_ENCRYPT authentication.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

dftdbpath - Default database path

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable Online

Propagation Class Immediate

Default [Range]

UNIX	Home directory of instance owner [any existing path]
Windows	Drive on which DB2 is installed [any existing path]

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the *dftdbpath* parameter.

In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path (on UNIX-based platforms), or a network drive (in a Windows environment). The specified path must physically exist on each database partition server. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the node name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For UNIX-based environments, the length of the *dftdbpath* name cannot exceed 215 characters and must be a valid, absolute, path name. For Windows, the *dftdbpath* can be a drive letter, optionally followed by a colon.

Recommendation: If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

fed_noauth - Bypass federated authentication

Configuration Type Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable online

Propagation Class Immediate

Default [Range] No [Yes; No]

When *fed_noauth* is set to *yes*, *authentication* is set to *server* or *server_encrypt*, and *federated* is set to *yes*, then authentication at the instance is bypassed. It is assumed that authentication will happen at the data source. Exercise caution when *fed_noauth* is set to *yes*. Authentication is done at neither the client nor at DB2. Any user who knows the SYSADM authentication name can assume SYSADM authority for the federated server.

Related reference:

- "authentication - Authentication type " on page 506
- "federated - Federated database system support " on page 501
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

group_plugin - Group plug-in

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the group plug-in library. By default, this value is null, and DB2 uses the operating system group lookup. The plug-in will be used for all group lookups.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

local_gssplugin - GSS API plug-in used for local instance level authorization

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the default GSS API plug-in library to be used for instance level local authorization when the value of the *authentication* database manager configuration parameter is set to GSSPLUGIN or GSS_SERVER_ENCRYPT.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

srvcon_auth - Authentication type for incoming connections at the server

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [CLIENT; SERVER; SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

This parameter specifies how and where user authentication is to take place when handling incoming connections at the server; it is used to override the current authentication type. If a value is not specified, DB2 uses the value of the *authentication* database manager configuration parameter.

For a description of each authentication type, see "authentication - Authentication type " on page 506.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the GSS API plug-in libraries that are supported by the database server. By default, the value is null. If the authentication type is GSSPLUGIN and this parameter is NULL, an error is returned. If the authentication type is KERBEROS and this parameter is NULL, the DB2-supplied kerberos module or library is used. This parameter is not used if another authentication type is used.

When the authentication type is KERBEROS and the value of this parameter is not NULL, the list must contain exactly one Kerberos plug-in, and that plug-in is used for authentication (all other GSS plug-ins in the list are ignored). If there is more than one Kerberos plug-in, an error is returned.

Each GSS API plug-in name must be separated by a comma (,) with no space either before or after the comma. Plug-in names should be listed in the order of preference. This parameter handles incoming connections at the server when the *srvcon_auth* parameter is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT, or when *srvcon_auth* is not specified, and authentication is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] Null [any valid string]

This parameter specifies the name of the default userid-password plug-in library to be used for server-side authentication. By default, the value is null and the DB2-supplied userid-password plug-in library is used.

The parameter handles incoming connections at the server when the *srvcon_auth* parameter is specified as SERVER or SERVER_ENCRYPT, or when *srvcon_auth* is not specified, and *authentication* is specified as CLIENT, SERVER, or SERVER_ENCRYPT.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

srv_plugin_mode - Server plug-in mode

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] UNFENCED

This parameter specifies whether plug-ins are to run in fenced mode or unfenced mode. Unfenced mode is the only supported mode.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

sysadm_group - System administration authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients

- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

System administration (SYSADM) authority is the highest level of authority within the database manager and controls all database objects. This parameter defines the group name with SYSADM authority for the database manager instance.

SYSADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating system, this parameter can be set to any local group, and is defined in the Windows 2000 security database. Group names must be 30 bytes or less in length. If "NULL" is specified for this parameter, all members of the Administrators group have SYSADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the SYSADM group can be any valid UNIX group name.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSADM_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "sysctrl_group - System control authority group name " on page 515
- "sysmaint_group - System maintenance authority group name " on page 516
- "sysmon_group - System monitor authority group name " on page 517

sysctrl_group - System control authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but does not allow direct access to data.

Group names on all platforms are accepted as long as they are 30 bytes or less in length.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSCTRL group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSCTRL_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "sysadm_group - System administration authority group name " on page 514
- "sysmaint_group - System maintenance authority group name " on page 516
- "sysmon_group - System monitor authority group name " on page 517

sysmaint_group - System maintenance authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter defines the group name with system maintenance (SYSMAINT) authority. SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Group names on all platforms are accepted as long as they are 30 bytes or less in length.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMAINT group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMAINT_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "sysadm_group - System administration authority group name " on page 514
- "sysctrl_group - System control authority group name " on page 515
- "sysmon_group - System monitor authority group name " on page 517

sysmon_group - System monitor authority group name

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Client
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default Null

This parameter defines the group name with system monitor (SYSMON) authority. Users having SYSMON authority at the instance level have the ability to take database system monitor snapshots of a database manager instance or its databases. SYSMON authority includes the ability to use the following commands:

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

Users with SYSADM, SYSCTRL, or SYSMAINT authority automatically have the ability to take database system monitor snapshots and to use these commands.

Group names on all platforms are accepted as long as they are 30 bytes or less in length.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not

store group information, thereby providing no way of determining if a user is a member of a designated SYSMON group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMON_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "sysadm_group - System administration authority group name " on page 514
- "sysctrl_group - System control authority group name " on page 515
- "sysmaint_group - System maintenance authority group name " on page 516

trust_allclnts - Trust all clients

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	YES [NO, YES, DRDAONLY]

This parameter is only active when the *authentication* parameter is set to CLIENT.

This parameter and *trust_clntauth* are used to determine where users are validated to the database environment.

By accepting the default of "YES" for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to "NO" if the *authentication* parameter is set to CLIENT. If this parameter is set to "NO", the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

Setting this parameter to "DRDAONLY" protects against all clients except clients from DB2 for OS/390 and z/OS®, DB2 for VM and VSE, and DB2 for OS/400®. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

When *trust_allclnts* is set to "DRDAONLY", the *trust_clntauth* parameter is used to determine where the clients are authenticated. If *trust_clntauth* is set to "CLIENT",

authentication occurs at the client. If *trust_clntauth* is set to "SERVER", authentication occurs at the client if no password is provided, and at the server if a password is provided.

Related reference:

- "authentication - Authentication type " on page 506
- "trust_clntauth - Trusted clients authentication " on page 519
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

trust_clntauth - Trusted clients authentication

Configuration Type Database manager

Applies to

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients

Parameter Type Configurable

Default [Range] CLIENT [CLIENT, SERVER]

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection. This parameter (and *trust_allclnts*) is only active if the *authentication* parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

If this parameter is set to CLIENT (the default), the trusted client can connect without providing a user ID and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to SERVER, the user ID and password will be validated at the server.

The numeric value for CLIENT is 0. The numeric value for SERVER is 1.

Related reference:

- "authentication - Authentication type " on page 506
- "trust_allclnts - Trust all clients " on page 518
- "GET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION command" in *Command Reference*

restrict_access - Database has restricted access configuration parameter

Configuration Type Database

Parameter Type Informational

This parameter indicates whether the database was created using the restrictive set of default actions. In other words if it was created with the RESTRICTIVE clause in the CREATE DATABASE command.

YES The RESTRICTIVE clause was used in the CREATE DATABASE command when this database was created.

NO The RESTRICTIVE clause was not used in the CREATE DATABASE command when this database was created.

DB2 Administration Server

The following parameters relate to the DB2 administration server:

authentication - Authentication type DAS

Configuration Type DB2 Administration Server

Applies to DB2 Administration Server

Parameter Type Configurable

Default [Range] SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

This parameter determines how and where authentication of a user takes place.

If authentication is SERVER_ENCRYPT, then the user ID and password are sent from the client to the server so authentication can take place on the server. Passwords sent over the network are encrypted.

A value of KERBEROS_ENCRYPT means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication.

Note: The KERBEROS_ENCRYPT authentication type is only supported on servers running Windows 2000.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

contact_host - Location of contact list

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid Version 8 DB2 administration server TCP/IP hostname]

This parameter specifies the location where the contact information used for notification by the Scheduler and the Health Monitor is stored. The location is defined to be a DB2 administration server's TCP/IP hostname. Allowing *contact_host* to be located on a remote DAS provides support for sharing a contact list across multiple DB2 administration servers. If *contact_host* is not specified, the DAS assumes the contact information is local.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

das_codepage - DAS code page

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid DB2 code page]

This parameter indicates the code page used by the DB2 administration server. If the parameter is null, then the default code page of the system is used. This parameter should be compatible with the locale of the local DB2 instances. Otherwise, the DB2 administration server cannot communicate with the DB2 instances.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "das_territory - DAS territory " on page 522
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

das_territory - DAS territory

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid DB2 territory]

This parameter shows the territory used by the DB2 administration server. If the parameter is null, then the default territory of the system is used.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "das_codepage - DAS code page " on page 521
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

dasadm_group - DAS administration authority group name

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid group name]

DAS Administration (DASADM) authority is the highest level of authority within the DAS. This parameter defines the group name with DASADM authority for the DAS.

DASADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating systems, this parameter can be set to any local group that is defined in the Windows security database. Group names are accepted as long as they are 30 bytes or less in length.If "NULL" is specified for this parameter, all members of the Administrators group have DASADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the DASADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the DASADM group can be any valid UNIX group name.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

db2system - Name of the DB2 server system

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Default [Range]	TCP/IP host name [any valid system name]

This parameter specifies the name that is used by your users and database administrators to identify the DB2 server system. If possible, this name should be unique within your network.

This name is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center.

When using the 'Search the Network' function of the Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. This name aids users in identifying the system that contains the database they wish to access. A value for *db2system* is set at installation time as follows:

- On Windows, the setup program sets it equal to the computer name specified for the Windows system.
- On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.

Related reference:

- "discover - DAS discovery mode " on page 523
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

discover - DAS discovery mode

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	SEARCH [DISABLE; KNOWN; SEARCH]

From an administration server perspective, this configuration parameter determines the type of discovery mode that is started when the DB2 Administration Server starts.

- If discover = SEARCH, the administration server handles SEARCH discovery requests from clients. SEARCH provides a superset of the functionality provided by KNOWN discovery. When discover = SEARCH, the administration server will handle both SEARCH and KNOWN discovery requests from clients.

- If discover = KNOWN, the administration server handles only KNOWN discovery requests from clients.
- If discover = DISABLE, then the administration server will not handle any type of discovery request. The information for this server system is essentially hidden from clients.

The default discovery mode is SEARCH.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:
- "db2system - Name of the DB2 server system " on page 523
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

exec_exp_task - Execute expired tasks

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	No [Yes; No]

This parameter specifies whether or not the Scheduler will execute tasks that have been scheduled in the past, but have not yet been executed. The Scheduler only detects expired tasks when it starts up.

For example, if you have a job scheduled to run every Saturday, and the Scheduler is turned off on Friday and then restarted on Monday, the job scheduled for Saturday is now a job that is scheduled in the past. If *exec_exp_task* is set to Yes, your Saturday job will run when the Scheduler is restarted.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:
- "smtp_server - SMTP server " on page 527
- "sched_enable - Scheduler mode " on page 526
- "sched_userid - Scheduler user ID " on page 526
- "toolscat_db - Tools catalog database " on page 527
- "toolscat_inst - Tools catalog database instance " on page 528
- "toolscat_schema - Tools catalog database schema " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

jdk_64_path - 64-Bit Software Developer's Kit for Java installation path DAS

Configuration Type	DB2 Administration Server

Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid path]

This parameter specifies the directory under which the 64-Bit Software Developer's Kit (SDK) for Java, to be used for running DB2 administration server functions, is installed.

Note: This is different from the *jdk_path* configuration parameter, which specifies a 32-bit SDK for Java.

Environment variables used by the Java interpreter are computed from the value of this parameter. This parameter is only used on those platforms that support both 32- and 64-bit instances.

Those platforms are:

- 64-bit kernels of AIX, HP-UX, and Solaris operating systems
- 64-bit Windows on X64 and IPF
- 64-bit Linux kernel on x86-64, POWER(TM), and zSeries(R).

On all other platforms, only *jdk_path* is used.

Because there is no default value for this parameter, you should specify a value when you install the SDK for Java.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

jdk_path - Software Developer's Kit for Java installation path DAS

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Default Java install path [any valid path]

This parameter specifies the directory under which the Software Developer's Kit (SDK) for Java, to be used for running DB2 administration server functions, is installed. Environment variables used by the Java interpreter are computed from the value of this parameter.

On Windows operating systems, Java files (if needed) are placed under the sqllib directory (in java\jdk) during DB2 installation. The *jdk_path* configuration parameter is then set to sqllib\java\jdk. Java is never actually installed by DB2 on

Windows platforms; the files are merely placed under the sqllib directory, and this is done regardless of whether or not Java is already installed.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

sched_enable - Scheduler mode

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Off [On; Off]

This parameter indicates whether or not the Scheduler is started by the administration server. The Scheduler allows tools such as the Task Center to schedule and execute tasks at the administration server.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "exec_exp_task - Execute expired tasks " on page 524
- "sched_userid - Scheduler user ID " on page 526
- "smtp_server - SMTP server " on page 527
- "toolscat_db - Tools catalog database " on page 527
- "toolscat_inst - Tools catalog database instance " on page 528
- "toolscat_schema - Tools catalog database schema " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

sched_userid - Scheduler user ID

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Informational
Default [Range]	Null [any valid user ID]

This parameter specifies the user ID used by the Scheduler to connect to the tools catalog database. This parameter is only relevant if the tools catalog database is remote to the DB2 administration server.

The userid and password used by the Scheduler to connect to the remote tools catalog database are specified using the **db2admin** command.

Related reference:

- "exec_exp_task - Execute expired tasks " on page 524
- "sched_enable - Scheduler mode " on page 526
- "smtp_server - SMTP server " on page 527
- "toolscat_db - Tools catalog database " on page 527
- "toolscat_inst - Tools catalog database instance " on page 528
- "toolscat_schema - Tools catalog database schema " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*

smtp_server - SMTP server

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	Null [any valid SMTP server TCP/IP hostname]

This parameter is used by the Scheduler and the Health Monitor.

When the Scheduler is on, this parameter identifies the SMTP server that the Scheduler will use to send e-mail and pager notifications.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "exec_exp_task - Execute expired tasks " on page 524
- "sched_enable - Scheduler mode " on page 526
- "sched_userid - Scheduler user ID " on page 526
- "toolscat_db - Tools catalog database " on page 527
- "toolscat_inst - Tools catalog database instance " on page 528
- "toolscat_schema - Tools catalog database schema " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

toolscat_db - Tools catalog database

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid database alias]

This parameter indicates the tools catalog database used by the Scheduler. This database must be in the database directory of the instance specified by *toolscat_inst*.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "exec_exp_task - Execute expired tasks " on page 524
- "sched_enable - Scheduler mode " on page 526
- "sched_userid - Scheduler user ID " on page 526
- "smtp_server - SMTP server " on page 527
- "toolscat_inst - Tools catalog database instance " on page 528
- "toolscat_schema - Tools catalog database schema " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

toolscat_inst - Tools catalog database instance

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid instance]

This parameter indicates the instance name that is used by the Scheduler, along with toolscat_db and toolscat_schema, to identify the tools catalog database. The tools catalog database contains task information created by the Task Center and the Control Center. The tools catalog database must be listed in the database directory of the instance specified by this configuration parameter. The database can be local or remote. If the tools catalog database is local, the instance must be configured for TCP/IP. If the database is remote, the node cataloged in the database directory must be a TCP/IP node.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related tasks:

- "Tools catalog database and DB2 administration server (DAS) scheduler setup and configuration" in *Administration Guide: Implementation*

Related reference:

- "exec_exp_task - Execute expired tasks " on page 524
- "sched_enable - Scheduler mode " on page 526
- "sched_userid - Scheduler user ID " on page 526
- "smtp_server - SMTP server " on page 527
- "toolscat_db - Tools catalog database " on page 527
- "toolscat_schema - Tools catalog database schema " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

toolscat_schema - Tools catalog database schema

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server

Parameter Type	Configurable
Default [Range]	Null [any valid schema]

This parameter indicates the schema of the tools catalog database used by the Scheduler. The schema is used to uniquely identify a set of tools catalog tables and views within the database.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:

- "exec_exp_task - Execute expired tasks " on page 524
- "sched_enable - Scheduler mode " on page 526
- "sched_userid - Scheduler user ID " on page 526
- "smtp_server - SMTP server " on page 527
- "toolscat_db - Tools catalog database " on page 527
- "toolscat_inst - Tools catalog database instance " on page 528
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

Appendix B. DB2 Registry and Environment Variables

This chapter describes how to use the registry and environment variables and provides lists of the variables in each category with an explanation of their syntax and usage.

DB2 registry and environment variables

DB2 provides a number of registry variables and environment variables that you may need to know about to get up and running.

To view a list of all supported registry variables, execute the following command:

```
db2set -lr
```

To change the value for a variable in the current or default instance, execute the following command:

```
db2set registry_variable_name=new_value
```

Whether the DB2 environment variables DB2INSTANCE, DB2NODE, DB2PATH, and DB2INSTPROF are stored in the DB2 profile registries depends on your operating system. To update these environment variables, use the `set` command. These changes are only effective in the local (current) command Window and are in effect until the next time the system is rebooted. On UNIX platforms, you can use the `export` command instead of the `set` command.

You must set the values for the changed registry variables before you execute the DB2START command.

Note: If a registry variable requires Boolean values as arguments, the values YES, 1, and ON are all equivalent and the values NO, 0, and OFF are also equivalent. For any variable, you can specify any of the appropriate equivalent values.

Related concepts:

- "Environment variables and the profile registry" in *Administration Guide: Implementation*

Related tasks:

- "Setting DB2 registry variables at the user level in the LDAP environment" in *Administration Guide: Implementation*

Related reference:

- "Command-line variables" on page 549
- "Communications variables" on page 543
- "Miscellaneous variables" on page 572
- "MPP configuration variables" on page 550
- "Performance variables" on page 556
- "Query compiler variables" on page 551
- "System environment variables" on page 536

Registry and environment variables by category

The following sections list the registry and environment variables according to the aspect of database manager or database behavior that they control.

General registry variables

DB2ACCOUNT

- Operating system: All
- Default=null
- This variable defines the accounting string that is sent to the remote host. Refer to the *DB2 Connect User's Guide* for details.

DB2BIDI

- Operating system: All
- Default=NO, Values: YES or NO
- This variable enables bidirectional support and the DB2CODEPAGE variable is used to declare the code page to be used.

DB2CODEPAGE

- Operating system: All
- Default: derived from the language ID, as specified by the operating system.
- This variable specifies the code page of the data presented to DB2 for database client application. The user should not set DB2CODEPAGE unless explicitly stated in DB2 documents, or asked to do so by DB2 service. Setting DB2CODEPAGE to a value not supported by the operating system can produce unexpected results. Normally, you do not need to set DB2CODEPAGE because DB2 automatically derives the code page information from the operating system.

  Note: Because Windows does not report a Unicode code page (in the Windows regional settings) instead of the ANSII code page, a Windows application will not behave as a Unicode client. To override this behavior, set the DB2CODEPAGE registry variable to `1208` (for the Unicode code page) to cause the application to behave as a Unicode application.

DB2_COLLECT_TS_REC_INFO

- Operating system: All
- Default=ON, Values: ON or OFF
- This variable specifies whether DB2 will process all log files when rolling forward a table space, regardless of whether the log files contain log records that affect the table space. To skip the log files known not to contain any log records affecting the table space, set this variable to "ON". DB2_COLLECT_TS_REC_INFO must be set before the log files are created and used so that the information required for skipping log files is collected.

DB2_CONNRETRIES_INTERVAL

- Operating system: All
- Default= not set, Values: an integer number of seconds
- This variable specifies the sleep time between consecutive connection retries, in seconds, for the automatic client reroute feature. You can use

this variable in conjunction with DB2_MAX_CLIENT CONNRETRIES to configure the retry behavior for automatic client reroute.

If DB2_MAX_CLIENT_CONNRETRIES is set, but DB2_CONNRETRIES_INTERVAL is not, DB2_CONNRETRIES_INTERVAL defaults to 30. If DB2_MAX_CLIENT_CONNRETRIES is not set, but DB2_CONNRETRIES_INTERVAL is set, DB2_MAX_CLIENT_CONNRETRIES defaults to 10. If neither DB2_MAX_CLIENT_CONNRETRIES nor DB2_CONNRETRIES_INTERVAL is set, the automatic client reroute feature reverts to it's default behavior of retrying the connection to a database repeatedly for up to 10 minutes.

DB2CONSOLECP

- Operating system: Windows
- Default= null, Values: all valid code page values
- Specifies the codepage for displaying DB2 message text. When specified, this value overrides the operating system codepage setting.

DB2COUNTRY

- Operating system: Windows
- Default=null, Values: all valid numeric country, territory, or region codes
- This variable specifies the country, territory, or region code of the client application. When specified, this value overrides the operating system setting.

DB2DBDFT

- Operating system: All
- Default=null
- This variable specifies the database alias name of the database to be used for implicit connects. If an application has no database connection but SQL or XQuery statements are issued, an implicit connect will be made if the DB2DBDFT environment variable has been defined with a default database.

DB2DBMSADDR

- Operating system: Windows 32-bit
- Default= 0x20000000, Values: 0x20000000 to 0xB0000000 in increments of 0x10000
- This variable specifies the default database manager shared memory address in hexadecimal format. If *db2start* fails due to a shared memory address collision, this registry variable can be modified to force the database manager instance to allocate its shared memory at a different address.

DB2DISCOVERYTIME

- Operating system: Windows
- Default=40 seconds, Minimum=20 seconds
- This variable specifies the amount of time that SEARCH discovery will search for DB2 systems.

DB2FFDC

- Operating system: All
- Default: ON, Values: ON, CORE:OFF

- Provides the ability to deactivate core file generation. By default, this registry variable is set to `ON`. If this registry variable is not set, or is set to a value other than `CORE:OFF`, core files may be generated if the DB2 server abends. (Core files are used for problem determination, and are created in the DIAGPATH.)

  Note: On Linux platforms, the default core file size limit is set to to `0` (that is, `ulimit -c`). With this setting, core files are not generated. To allow core files to be created on Linux platforms, set the value to unlimited.

  Core files contain the entire process image of the terminating DB2 process. Consideration should be given to the available file system space as core files can be quite large. The size is dependent on the DB2 configuration and the state of the process at the time the problem occurs.

DB2_FORCE_APP_ON_MAX_LOG

- Operating system: All
- Default: TRUE, Values: TRUE, FALSE
- Specifies what happens when the MAX_LOG configuration parameter value is exceeded. If set to TRUE, the application is forced off the database and the unit of work is rolled back.

  If FALSE, the current statement fails. The application can still commit the work completed by previous statements in the unit of work, or it can roll back the work completed to undo the unit of work.

DB2GRAPHICUNICODESERVER

- Operating system: All
- Default=OFF, Values: ON or OFF
- This registry variable is used to accommodate existing applications written to insert graphic data into a Unicode database. Its use is only needed for applications that specifically send sqldbchar (graphic) data in Unicode instead of the code page of the client. (sqldbchar is a supported SQL data type in C and C++ that can hold a single double-byte character.) When set to "ON", you are telling the database that graphic data is coming in Unicode, and the application expects to receive graphic data in Unicode.

DB2INCLUDE

- Operating system: All
- Default=current directory
- Specifies a path to be used during the processing of the SQL INCLUDE text-file statement during DB2 PREP processing. It provides a list of directories where the INCLUDE file might be found. Refer to *Developing Embedded SQL Applications* for descriptions of how DB2INCLUDE is used in the different precompiled languages.

DB2INSTDEF

- Operating system: Windows
- Default=DB2
- This variable sets the value to be used if DB2INSTANCE is not defined.

DB2INSTOWNER

- Operating system: Windows
- Default=null

- The registry variable created in the DB2 profile registry when the instance is first created. This variable is set to the name of the instance-owning machine.

DB2_LIC_STAT_SIZE

- Operating system: All
- Default=null, Range: 0 to 32 767
- This variable determines the maximum size (in MBs) of the file containing the license statistics for the system. A value of zero turns the license statistic gathering off. If the variable is not recognized or not defined, the variable defaults to unlimited. The statistics are displayed using the License Center.

DB2LOCALE

- Operating system: All
- Default= NO, Values: YES or NO
- This variable specifies whether the default ″C″ locale of a process is restored to the default ″C″ locale after calling DB2 and whether to restore the process locale back to the original 'C' after calling a DB2 function. If the original locale was not 'C', then this registry variable is ignored.

DB2_MAX_CLIENT_CONNRETRIES

- Operating system: All
- Default=not set, Values: an integer number of maximum times to retry the connection
- This variable specifies the maximum number of connection retries that the automatic client reroute feature will attempt. You can use this variable in conjunction with DB2_CONNRETRIES_INTERVAL to configure the retry behavior for automatic client reroute.

  If DB2_MAX_CLIENT_CONNRETRIES is set, but DB2_CONNRETRIES_INTERVAL is not, DB2_CONNRETRIES_INTERVAL defaults to 30. If DB2_MAX_CLIENT_CONNRETRIES is not set, but DB2_CONNRETRIES_INTERVAL is set, DB2_MAX_CLIENT_CONNRETRIES defaults to 10. If neither DB2_MAX_CLIENT_CONNRETRIES nor DB2_CONNRETRIES_INTERVAL is set, the automatic client reroute feature reverts to it's default behavior of retrying the connection to a database repeatedly for up to 10 minutes.

DB2NBDISCOVERRCVBUFS

- Operating system: All
- Default=16 buffers, Minimum=16 buffers
- This variable is used for NetBIOS search discovery. The variable specifies the number of concurrent discovery responses that can be received by a client. If the client receives more concurrent responses than are specified by this variable, then the excess responses are discarded by the NetBIOS layer. The default is sixteen (16) NetBIOS receive buffers. If a number less than the default value is chosen, then the default is used.

DB2_OBJECT_TABLE_ENTRIES

- Operating system: All
- Default=0, Values: 0–65532

The actual maximum value possible on your system depends on the page size and extent size, but it cannot exceed 65532.

- This variable specifies the expected number of objects in a table space. If you know that a large number of objects (for example, 1000 or more) will be created in a DMS table space, you should set this registry variable to the approximate number before creating the table space. This will reserve contiguous storage for object metadata during table space creation. Reserving contiguous storage reduces the chance that an online backup will block operations which update entries in the metadata (for example, CREATE INDEX, IMPORT REPLACE). It will also make resizing the table space easier because the metadata will be stored at the start of the table space.

  If the initial size of the table space is not large enough to reserve the contiguous storage, the table space creation will continue without the additional space reserved.

DB2TERRITORY

- Operating system: All
- Default: derived from the language ID, as specified by the operating system.
- This variable specifies the region, or territory code of the client application, which influences date and time formats.

DB2_VIEW_REOPT_VALUES

- Operating system: All
- Default=NO, Values: YES, NO
- This variable enables all users to store the cached values of a reoptimized SQL or XQuery statement in the EXPLAIN_PREDICATE table when the statement is explained. When this variable is set to NO, only DBADM is allowed to save these values in the EXPLAIN_PREDICATE table.

Related concepts:

- "DB2 registry and environment variables" on page 531

System environment variables

DB2_ALTERNATE_GROUP_LOOKUP

- Operating system: AIX
- Default: NULL, Values: NULL or GETGRSET
- This variable allows DB2 to obtain group information from an alternative source provided by the operating system. On AIX, the function getgrset is used. This provides the ability to obtain groups from somewhere other than local files via Loadable Authentication Modules.

DB2_CLP_EDITOR

- Operating system: All
- Default: Windows platforms: 'Notepad', UNIX: 'vi', Values: Any valid editor that is located in the operating system path

  Note: This registry variable is not set to the default value during installation. Instead, the code that makes use of this variable uses a default value if the registry variable is not set.

- This variable determines the editor to be used when executing the **EDIT** command. From a CLP interactive session, the **EDIT** command launches an editor pre-loaded with a user-specified command which can then be edited and run.

DB2_CLPHISTSIZE

- Operating system: All
- Default: 20, Values: 1–500 inclusive

  Note: This registry variable is not set to the default value during installation. Instead, the code that makes use of this variable uses a default value of 20 if the registry variable is not set or if it is set to a value outside of the valid range.
- This variable determines the number of commands stored in the command history during CLP interactive sessions. Because the command history is held in memory, a very high value for this variable might result in a performance impact depending on the number and length of commands run in a session.

DB2CONNECT_IN_APP_PROCESS

- Operating system: All
- Default=YES, Values: YES or NO
- When you set this variable to NO, local DB2 Connect clients on a DB2 Connect Enterprise Edition machine are forced to run within an agent. Some advantages of running within an agent are that local clients can be monitored and that they can use SYSPLEX support.

DB2_COPY_NAME

- Operating system: Windows
- Default=the name of the default copy of DB2 installed on your machine. Values: the name of a copy of DB2 installed on your machine. The name can be up to 128 characters long.
- The DB2_COPY_NAME variable stores the name of the copy of DB2 currently in use. If you have multiple DB2 copies installed on your machine, you cannot use DB2_COPY_NAME to switch to a different copy of DB2, you must run the command [INSTALLPATH]\bin\ db2envars.bat to change the copy currently in use.

DB2DBMSADDR

- Operating system: Linux on x86 and Linux on zSeries® (31-bit)
- Default: null, Values: virtual addresses in the range 0x09000000 to 0xB0000000 in increments of 0x10000
- The DB2DBMSADDR registry variable specifies the default database shared memory address in hexadecimal format.

  Note: An incorrect address can cause severe issues with the DB2 system, ranging from an inability to start a DB2 instance, to an ability to connect to the database. An incorrect address is one that collides with an area in memory that is already in use, or is predestined to be used for something else. To address this problem, reset the DB2DBMSADDR registry variable to null by using the following command:

  ```
  db2set DB2DBMSADDR=
  ```

  This variable can be set in conjunction with DB2_MAPPED_BASE, or alone, to fine tune the address space layout of DB2 processes. This

variable changes the location of the instance shared memory from its current location at virtual address 0x10000000 to the new value.

DB2DOMAINLIST

- Operating system: Windows server only
- Default=null, Values: A list of Windows domain names separated by commas (“,”).
- This variable defines one or more Windows domains. The list defines the domains that the requesting user ID will be authenticated against. Only users belonging to these domains will have their connection or attachment requests accepted.

  This variable is effective only when CLIENT authentication is set in the Database Manager configuration and is needed if a single sign on from a Windows desktop is required in a Windows domain environment.

  This registry variable should only be used under a pure Windows domain environment with DB2 servers and clients running DB2 Universal Database™ Version 7.1 (or later).

DB2ENVLIST

- Operating system: UNIX
- Default: null
- This variable lists specific variable names for either stored procedures or user-defined functions. By default, the **db2start** command filters out all user environment variables except those prefixed with **DB2** or **db2**. If specific environment variables must be passed to either stored procedures or user-defined functions, you can list the variable names in the DB2ENVLIST environment variable. Separate each variable name by one or more spaces.

DB2INSTANCE

- Operating system: All
- Default=DB2INSTDEF on Windows 32-bit operating systems.
- The environment variable used to specify the instance that is active by default. On UNIX, users must specify a value for DB2INSTANCE.

DB2INSTPROF

- Operating system: Windows
- Default: null
- The environment variable used to specify the location of the instance directory on Windows operating systems, if different than DB2PATH.

DB2LIBPATH

- Operating system: UNIX
- Default: null
- DB2 constructs its own shared library path. If you want to add a PATH into the engine’s library path (for example, on AIX, a user-defined function requires a specific entry in LIBPATH), you must set DB2LIBPATH. The actual value of DB2LIBPATH is appended to the end of the DB2 constructed shared library path.

DB2LOGINRESTRICTIONS

- Operating system: AIX
- Default: LOCAL, Values: LOCAL, REMOTE, SU, NONE

- This registry variable allows you to use an AIX operating system API called loginrestrictions(). This API determines whether a user is allowed to access the system. By calling this API, DB2 database security is able to enforce the login restrictions that are specified by the operating system. There are different values that can be submitted to this API when using this registry variable. The values are:
  - REMOTE

    DB2 only enforces login restrictions to verify that the account can be used for remote logins through the *rlogind* or *telnetd* programs.
  - SU

    DB2 V9.1 only enforces su restrictions to verify that the "su" command is permitted, and that the current process has a group ID that can invoke the "su" command to switch to the account.
  - NONE

    DB2 does not enforce any login restrictions.
  - LOCAL (or the variable is not set)

    DB2 only enforces login restrictions to verify that local logins are permitted for this account. This is the normal behavior when logging in.

  No matter which one of these options you set, user accounts or IDs that have the specified privileges are able to use DB2 successfully both locally on the server and from remote clients. For a description of the loginrestrictions() API, please refer to AIX documentation.

DB2_MAPPED_BASE

- Operating system: Linux on x86 and Linux on zSeries (31-bit)
- Default: null, Values: 0, or (hex) virtual address in the 31-bit and 32-bit range, or null (not set).
- The DB2_MAPPED_BASE registry variable is used to increase the amount of contiguous virtual address space available to a DB2 process by relocating the attachment address of the shared libraries for the specific process. The contiguous virtual address space is important to maximize the amount of database shared memory available to the DB2 database system. This variable is only effective on distributions that include the mapped-base file in the process identification directory in the *proc* file system.

  DB2 attempts to relocate the shared libraries to the virtual address 0x20000000 if this variable is not set.

  The registry variable can also be set to any virtual address (in hex) in the range of the 31-bit and 32-bit address space if the new address places the shared libraries lower in the address space.

  Note: An incorrect address can cause severe issues with a DB2 database system, ranging from an inability to start DB2 database instances, to an ability to connect to the database. An incorrect address is one that collides with an area in memory that is already in use, or is predestined to be used for something else. To address this problem, reset the DB2_MAPPED_BASE registry variable to null by using the following command:

  ```
  db2set DB2_MAPPED_BASE=
  ```

  The following message may appear multiple times in the db2diag.log file because this change is required once per logical note:

```
ADM0506I  DB2 has automatically updated the "mapped_base"
kernel parameter from "0x40000000(hex) 1073741824 (dec)" to
the recommended value "0x20000000(hex) 536870912 (dec)"
```

This message will only appear if setting of the registry variable is successful. The message includes the address where the shared files are located.

DB2NODE

- Operating system: All
- Default: null, Values: 1 to 999
- Used to specify the target logical node of a DB2 Enterprise Server Edition database partition server that you want to attach to or connect to. If this variable is not set, the target logical node defaults to the logical node which is defined with port `0` on the machine.

DB2OPTIONS

- Operating system: All
- Default: null
- Used to set the command line processor options.

DB2_PARALLEL_IO

- Operating system: All
- Default: null, Values: *TablespaceID*[:*n*],... – a comma-separated list of defined table spaces (identified by their numeric table space ID). If the prefetch size of a table space is AUTOMATIC, you can indicate to the DB2 database manager the number of disks per container for that table space by specifying the table space ID, followed by a colon, followed by the number of disks per container, *n*. If *n* is not specified, the default is 6.

  You can replace TablespaceID with an asterisk (*) to specify all table spaces. For example, if DB2_PARALLEL_IO=*, all table spaces will use six as the number of disks per container. If you specify both an asterisk (*) and a table space ID, the table space ID setting takes precedence. For example, if DB2_PARALLEL_IO=*,1:3, all table spaces will use six as the number of disks per container, except for table space 1, which will use three.
- This registry variable is used to change the way DB2 calculates the I/O parallelism of a table space. When I/O parallelism is enabled (either implicitly, by the use of multiple containers, or explicitly, by setting DB2_PARALLEL_IO), it is achieved by issuing the correct number of prefetch requests. Each prefetch request is a request for an extent of pages.

  If this registry variable is not set, the degree of parallelism of any table space is the number of containers of the table space. For example, if DB2_PARALLEL_IO is set to null and a table space has four containers, there will be four extent-sized prefetch requests issued.

  If this registry variable is set, and the prefetch size of the table is not AUTOMATIC, the degree of parallelism of the table space is the prefetch size divided by the extent size. For example, if DB2_PARALLEL_IO is set for a table space that has a prefetch size of 160 and an extent size of 32 pages, there will be five extent-sized prefetch requests issued.

  If this registry variable is set, and the prefetch size of the table space is AUTOMATIC, DB2 automatically calculates the prefetch size of a table space using the following equation:

```
Prefetch size =
 (number of containers) * (number of disks per container)
  * extent size
```

The number after the colon is used by DB2 for the *number of disks per container* in the equation. If only an asterisk is used but a number is not specified, a default of 6 disks per container is used.

The following table summarizes the different options available and how parallelism is calculated for each situation:

Table 76. How Parellism is Calculated

Prefetch size of table space	**DB2_PARALLEL_IO Setting**	**Parallelism is equal to:**
AUTOMATIC (Prefetchsize=number of containers * 1 * extent size)	Not set	Number of containers
AUTOMATIC (Prefetchsize=number of containers * 6 * extent size)	*Table space ID*	Number of containers * 6
AUTOMATIC (Prefetchsize=number of containers * *n* * extent size)	*Table space ID*:*n*	Number of containers * n
Not AUTOMATIC	Not set	Number of containers
Not AUTOMATIC	*Table space ID*	Prefetch size/extent size
Not AUTOMATIC	*Table space ID*:*n*	Prefetch size/extent size

For example, consider you have three table spaces and their ID's are 3,4, and 5 respectively. Their extent sizes are all 4096 bytes, and they all have two containers each. The prefetch sizes of table spaces 3 and 4 are both AUTOMATIC, and that of table space 5 is 16384 bytes. Suppose you set DB2_PARALLEL_IO=*:5,4:10, the table spaces will have parallelism derived as follows:

- Tablespace 3: The value of n (number of disks per container) is 5, extent size=4096, number of containers=2, and prefetch size is AUTOMATIC. Therefore, prefetch size is 2*5*4096 and parallelism=number of containers * n =2*5=10.
- Tablespace 4: Note the value of n (number of disks per container) is set specifically to 10 for this tablespace. Extent size=4096, number of containers=2 n=10, and prefetch size is AUTOMATIC. Therefore, prefetch size =2*10*4096, and parallelism=number of containers*n=2*10=20.
- Tablespace 5: The value of n is still 5, but it has no influence, as prefetch size is not AUTOMATIC. Extent size=4096 number of containers=2 and prefetch size=16384. Therefore, parallelism=prefetch size/extent size=16384/4096=4.

DB2PATH

- Operating system: Windows
- Default: (varies by operating system)
- This environment variable is used to specify the directory where the product is installed on Windows 32-bit operating systems.

DB2PROCESSORS

- Operating system: Windows
- Default: null, Values: 0–n-1 (where n= the number of processors)

- This variable sets the process affinity mask for a particular db2syscs process. In environments running multiple logical nodes, this variable is used to associate a logical node to a processor or set of processors.

  When specified, DB2 issues the SetProcessAffinityMask() api. If unspecified, the db2syscs process is associated with all processors on the server.

DB2RCMD_LEGACY_MODE

- Operating system: Windows,
- Default=null, Values: YES, ON, TRUE, or 1, or NO, OFF, FALSE, or 0
- This variable allows users to enable or disable the DB2 Remote Command Service's enhanced security. To run the DB2 Remote Command Service in a secure manner, set DB2RCMD_LEGACY_MODE to NO, OFF, FALSE, 0, or null. To run in legacy mode (without enhanced security), set DB2RCMD_LEGACY_MODE to YES, ON, TRUE, or 1. The secure mode is only available if your domain controller is running Windows 2000 or later.

  Note: If DB2RCMD_LEGACY_MODE is set to YES, ON, TRUE, or 1, all requests sent to the DB2 Remote Command Service are processed under the context of the requestor. To facilitate this, you must allow either or both the machine and service logon account to impersonate the client by enabling the machine and service logon accounts at the domain controller.

  Note: If DB2RCMD_LEGACY_MODE is set to NO, OFF, FALSE, or 0, you must have SYSADM authority in order to have the DB2 Remote Command Service execute commands on your behalf.

DB2SYSTEM

- Operating system: Windows and UNIX
- Default=null,
- Specifies the name that is used by your users and database administrators to identify the DB2 server system. If possible, this name should be unique within your network.

  This name is displayed in the system level of the Control Center's object tree to aid administrators in the identification of server systems that can be administered from the Control Center.

  When using the 'Search the Network' function of the Client Configuration Assistant, DB2 discovery returns this name and it is displayed at the system level in the resulting object tree. This name aids users in identifying the system that contains the database they wish to access. A value for DB2SYSTEM is set at installation time as follows:
  - On Windows the setup program sets it equal to the computer name specified for the Windows system.
  - On UNIX systems, it is set equal to the UNIX system's TCP/IP hostname.

DB2_USE_PAGE_CONTAINER_TAG

- Operating system: All
- Default: null, Values: ON, null
- By default, DB2 stores a container tag in the first extent of each DMS container, whether it is a file or a device. The container tag is the metadata for the container. Before DB2 Version 8.1, the container tag was

stored in a single page, and it thus required less space in the container. To continue to store the container tag in a single page, set DB2_USE_PAGE_CONTAINER_TAG to ON.

However, if you set this registry variable to ON when you use RAID devices for containers, I/O performance might degrade. Because for RAID devices you create table spaces with an extent size equal to or a multiple of the RAID stripe size, setting the DB2_USE_PAGE_CONTAINER_TAG to ON causes the extents not to line up with the RAID stripes. As a result, an I/O request might need to access more physical disks than would be optimal. Users are strongly advised against enabling this registry variable.

To activate changes to this registry variable, issue a DB2STOP command and then enter a DB2START command.

DB2_WORKLOAD

- Operating system: All
- Default: Not set, Values: Set to SAP to configure DB2 for SAP environment.
- Setting DB2_WORKLOAD to SAP automatically configures a number of DB2 registry variables for the SAP environment.

Related concepts:

- "DB2 registry and environment variables" on page 531

Related reference:

- "Command line processor options" in *Command Reference*

Communications variables

DB2CHECKCLIENTINTERVAL

- Operating system: All, server only
- Default=50, Values: A numeric value that is greater than or equal to zero.
- This variable specifies the frequency of TCP/IP client connection verifications. It permits early detection of client termination, instead of waiting until after the completion of the query. If this variable is set to 0, no verification is performed.

  Lower values cause more frequent checks. As a guide, for low frequency, use 100; for medium frequency, use 50; for high frequency use 10. The value is measured in an internal DB2 metric. The values represent a linear scale, that is, increasing the value from 50 to 100 doubles the interval. Checking more frequently for client status while executing a database request lengthens the time taken to complete queries. If the DB2 workload is heavy (that is, it involves many internal requests), setting DB2CHECKCLIENTINTERVAL to a low value has a greater impact on performance than in a situation where the workload is light.

  In DB2 Universal Database, Version 8.1.4, the default value for DB2CHECKCLIENTINTERVAL is 50. Prior to version 8.1.4, the default value was 0.

DB2COMM

- Operating system: All, server only
- Default=null, Values: NPIPE, TCPIP

- This variable specifies the communication managers that are started when the database manager is started. If this variable is not set, no DB2 communications managers are started at the server.

DB2_FORCE_NLS_CACHE

- Operating system: AIX, HP_UX, Solaris
- Default=FALSE, Values: TRUE or FALSE
- This variable is used to eliminate the chance of lock contention in multi-threaded applications. When this registry variable is "TRUE", the code page and territory code information is saved the first time a thread accesses it. From that point, the cached information is used for any other thread that requests this information. This eliminates lock contention and results in a performance benefit in certain situations. This setting should not be used if the application changes locale settings between connections. It is probably not needed in such a situation because multi-threaded applications typically do not change their locale settings because it is not "thread-safe" to do so.

DB2JD_PORT_NUMBER

- Operating system: All
- Default=6789, Values: 1-65535
- This variable can be set to override the default port number of the db2jd. If this registry variable is set, the db2jd will attempt to use it as the listening port of the db2jd when a db2jstrt is issued with no parameters. If this registry variable is not set, and no port parameter is provided, the db2jd will start on the default port of 6789.

DB2NBADAPTERS

- Operating system: Windows
- Default=0, Range: 0-15, multiple values should be separated by commas
- Used to specify which local adapters to use for DB2 NetBIOS LAN communications. Each local adapter is specified using its logical adapter number.

DB2NBCHECKUPTIME

- Operating system: Windows server only
- Default=1 minute, Values: 1-720
- Specifies the time interval between each invocation of the NetBIOS protocol checkup procedure. Checkup time is specified in minutes.

  Lower values ensure that the NetBIOS protocol checkup runs more often, freeing up memory and other system resources left when unexpected agent/session termination occurs.

DB2NBINTRLISTENS

- Operating system: Windows server only
- Default=1, Values: 1-10, multiple values should be separated by commas
- This variable specifies the number of NetBIOS listen send commands (NCBs) that are asynchronously issued in readiness for remote client interrupts. This flexibility is provided for ″interrupt active″ environments to ensure that interrupt calls from remote clients can establish connections when servers are busy servicing other remote interrupts.

  Setting DB2NBINTRLISTENS to a lower value conserves NetBIOS sessions and NCBs at the server. However, in an environment where

client interrupts are common, you may need to set DB2NBINTRLISTENS to a higher value in order to be responsive to interrupting clients.

Note: Values specified are position sensitive; they relate to the corresponding value positions for DB2NBADAPTERS.

DB2NBRECVBUFFSIZE

- Operating system: Windows server only
- Default=4096 bytes, Range: 4096-65536
- This variable specifies the size of the DB2 NetBIOS protocol receive buffers. These buffers are assigned to the NetBIOS receive NCBs. Lower values conserve server memory, while higher values may be required when client data transfers are larger.

DB2NBRECVNCBS

- Operating system: Windows server only
- Default=10, Range: 1-99
- This variable specifies the number of NetBIOS ″receive_any″ commands (NCBs) that the server issues and maintains during operation. This value is adjusted depending on the number of remote clients to which your server is connected. Lower values conserve server resources.

  Note: Each adapter in use can have its own unique receive NCB value specified by DB2NBRECVNCBS. The values specified are position sensitive; they relate to the corresponding value positions for DB2NBADAPTERS.

DB2NBRESOURCES

- Operating system: Windows server only
- Default=null
- Specifies the number of NetBIOS resources to allocate for DB2 use in a multi-context environment. This variable is restricted to multi-context client operation.

DB2NBSENDNCBS

- Operating system: Windows server only
- Default=6, Range: 1-720
- This variable specifies the number of send NetBIOS commands (NCBs) that the server reserves for use. This value can be adjusted depending on the number of remote clients your server is connected to. Setting DB2NBSENDNCBS to a lower value will conserve server resources. However, you might need to set it to a higher value to prevent the server from waiting to send to a remote client when all other send commands are in use.

DB2NBSESSIONS

- Operating system: Windows server only
- Default=null, Range: 5-254
- Specifies the number of sessions that DB2 should request to be reserved for DB2 use. The value of DB2NBSESSIONS can be set to request a specific session for each adapter specified using DB2NBADAPTERS.

  Note: Values specified are position sensitive; they relate to the corresponding value positions for DB2NBADAPTERS.

DB2NBXTRANCBS

- Operating system: Windows server only
- Default=5 per adapter, Range: 5-254
- Specifies the number of "extra" NetBIOS commands (NCBs) the server will need to reserve when the **db2start** command is issued. The value of DB2NBXTRANCBS can be set to request a specific session for each adapter specified using DB2NBADAPTERS.

DB2RETRY

- Operating system: Windows
- Default=0, Range: 0-20 000
- This variable sets the number of times DB2 attempts to restart the APPC listener. If the SNA subsystem at the server/gateway is down, this profile variable, in conjunction with DB2RETRYTIME, can be used to automatically restart the APPC listener without disrupting client communications using other protocols. In such a scenario, it is no longer necessary to stop and restart DB2 to reinstate the APPC client communications.

DB2RETRYTIME

- Operating system: Windows
- Default=1 minute, Range: 0-7 200 minutes
- In increments of one minute, the number of minutes that DB2 allows between performing successive retries to start the APPC listener. If the SNA subsystem at the server/gateway is down, this profile variable, in conjunction with DB2RETRY, can be used to automatically restart the APPC listener without disrupting client communications using other protocols. In such a scenario, it is no longer necessary to stop and restart DB2 to reinstate the APPC client communications.

DB2RSHCMD

- Operating system: UNIX
- Default=rsh (remsh on HP-UX), Values are a full path name to rsh, remsh, or ssh
- By default, DB2 database system uses rsh as the communication protocol when starting remote database partitions and with the db2_all script to run utilities and commands on all database partitions. For example, setting this registry variable to the full path name for ssh causes DB2 database products to use ssh as the communication protocol for the requested running of the utilities and commands. It may also be set to the full path name of a script that invokes the remote command program with appropriate default parameters. This variable is only required for partitioned databases, or for single-partition environments where the db2start command is run from a different server than where the DB2 product was installed. The instance owner must be able to use the specified remote shell program to log in from each DB2 database node to each other DB2 database node, without being prompted for any additional verification or authentication (that is, passwords or password phrases).

DB2RSHTIMEOUT

- Operating system: UNIX
- Default=30, Values are shown in seconds; accepted values are 1 to 120

- This variable is only applicable if DB2RSHCMD is set to a non-NULL value. This registry variable is used to control the timeout period that the DB2 database system will wait for any remote command. After this timeout period, if no response is received, the assumption is made that the remote database partition is not reachable and the operation has failed.

  Note: The time value given is not the time required to run the remote command, it is the time needed to authenticate the request.

DB2SERVICETPINSTANCE

- Operating system: Windows, AIX, Solaris
- Default=null
- Used to solve the problem caused by:
  - More than one instance running on the same machine
  - A Version 6 or Version 7 instance running on the same machine attempting to register the same TP names.

  When the **db2start** command is invoked, the instance specified will start the APPC listeners for the following TP names:
  - DB2DRDA
  - x'07'6DB

DB2SORCVBUF

- Operating system: All
- Default=65536
- Specifies the value of TCP/IP receive buffers on Windows operating systems.

DB2SOSNDBUF

- Operating system: All
- Default=65536
- Specifies the value of TCP/IP send buffers on Windows operating systems.

DB2SYSPLEX_SERVER

- Operating system: Windows, and UNIX
- Default=null
- Specifies whether SYSPLEX exploitation when connected to DB2 for OS/390 or z/OS is enabled. If this registry variable is not set (which is the default), or is set to a non-zero value, exploitation is enabled. If this registry variable is set to zero (0), exploitation is disabled. When set to zero, SYSPLEX exploitation is disabled for the gateway regardless of how the DCS database catalog entry has been specified. For more information, see the command-line processor **CATALOG DCS DATABASE** command.

DB2TCP_CLIENT_CONTIMEOUT

- Operating system: All, client only
- Default=0 (no timeout), Values: 0 to 32 767 seconds
- The DB2TCP_CLIENT_CONTIMEOUT registry variable specifies the number of seconds a client waits for the completion on a TCP/IP

connect operation. If a connection is not established in the seconds specified, then the DB2 database manager returns the error -30081 selectForConnectTimeout.

There is no timeout if the registry variable is not set or is set to 0.

Note: Operating systems also have a connection timeout value that may take effect prior to the timeout you set using DB2TCP_CLIENT_CONTIMEOUT. For example, AIX has a default tcp_keepinit=150 (in half seconds) that would terminate the connection after 75 seconds.

The DB2TCP_CLIENT_CONTIMEOUT variable is available starting with the DB2 database manager, version 9.1.

DB2TCP_CLIENT_RCVTIMEOUT

- Operating system: All, client only
- Default=0 (no timeout), Values: 0 to 32 767 seconds
- The DB2TCP_CLIENT_RCVTIMEOUT registry variable specifies the number of seconds a client waits for data on a TCP/IP receive operation. If data from the server is not received in the seconds specified, then the DB2 database manager returns the error -30081 selectForRecvTimeout.

  There is no timeout if the registry variable is not set or is set to 0.

  The DB2TCP_CLIENT_RCVTIMEOUT variable is available starting with the DB2 database manager, version 9.1.

DB2TCPCONNMGRS

- Operating system: All
- Default=1 on serial machines; square root of the number of processors rounded up to a maximum of sixteen connection managers on symmetric multiprocessor machines. Values: 1 to 16
- The default number of connection managers is created if the registry variable is not set. If the registry variable is set, the value assigned here overrides the default value. The number of TCP/IP connection managers specified up to a maximum of 16 is created. If less than one is specified then DB2TCPCONNMGRS is set to a value of one and a warning is logged that the value is out of range. If greater than sixteen is specified then DB2TCPCONNMGRS is set to a value of sixteen and a warning is logged that the value is out of range. Values between one and sixteen are used as given. When there is greater than one connection manager created, connection throughput should improve when multiple client connections are received simultaneously. There may be additional TCP/IP connection manager processes (on UNIX) or threads (on Windows operating systems) if the user is running on a SMP machine, or has modified the DB2TCPCONNMGRS registry variable. Additional processes or threads require additional storage.

  Note: Having the number of connection managers set to one causes a drop in performance on remote connections in systems with a lot of users, frequent connects and disconnects, or both.

Related concepts:

- "DB2 registry and environment variables" on page 531

Command-line variables

DB2BQTIME

- Operating system: All
- Default=1 second, Maximum value: 1 second
- This variable specifies the amount of time the command-line processor front end sleeps before it checks whether the back-end process is active and establishes a connection to it.

DB2BQTRY

- Operating system: All
- Default=60 retries, Minimum value: 0 retries
- This variable specifies the number of times the command-line processor front-end process tries to determine whether the back-end process is already active. It works in conjunction with DB2BQTIME.

DB2_CLPPROMPT

- Operating system: All
- Default=None (if it is not defined, "db2 => " will be used as the default CLP interactive prompt), Values: Any text string of length less than 100 that contains zero or more of the following tokens %i, %d, %ia, %da, or %n. Users need not set this variable unless they explicitly wish to change the default CLP interactive prompt (db2 =>).
- This registry variable allows a user to define the prompt to be used in the Command Line Processor (CLP) interactive mode. The variable can be set to any text string of length less than 100 characters containing zero or more of the optional tokens %i, %d, %ia, %da, or %n. When running in CLP interactive mode, the prompt to be used is constructed by taking the text-string specified in the DB2_CLPPROMPT registry variable and replacing all occurences of the tokens %i, %d, %ia, %da, or %n by the local alias of the current attached instance, the local alias of the current database connection, the authorization ID of the current attached instance, the authorization ID of the current database connection, and newline (that is, a carriage-return) respectively.

  Notes:

  1. If the DB2_CLPPROMPT registry variable is changed within CLP interactive mode, the new value for DB2_CLPPROMPT will not take effect until the CLP interactive mode has been closed and reopened.
  2. If no instance attachment exists, %ia is replaced by the empty string and %i is replaced by the value of the DB2INSTANCE registry variable. On Windows platforms only, if the DB2INSTANCE variable is not set, %i is replaced by the value of the DB2INSTDEF registry variable. If neither of these variables are set, %i is replaced by the empty string.
  3. If no database connection exists, %da is replaced by the empty string and %d is replaced by the value of the DB2DBDFT registry variable. If the DB2DBDFT variable is not set, %d is replaced by the empty string.
  4. The interactive input prompt will always present the values for the authorization IDs, database names, and instance names in upper case.

DB2IQTIME

- Operating system: All

- Default=5 seconds, Minimum value: 1 second
- Specifies the amount of time the command line processor back end process waits on the input queue for the front end process to pass commands.

DB2RQTIME

- Operating system: All
- Default=5 seconds, Minimum value: 1 second
- Specifies the amount of time the command line processor back end process waits for a request from the front end process.

Related concepts:

- "DB2 registry and environment variables" on page 531

MPP configuration variables

DB2CHGPWD_EEE

- Operating system: DB2 ESE on AIX and Windows
- Default=null, Values: YES or NO
- This variable specifies whether you allow other users to change passwords on AIX or Windows ESE systems. You must ensure that the passwords for all database partitions or nodes are maintained centrally using either a Windows domain controller on Windows, or NIS on AIX. If not maintained centrally, passwords may not be consistent across all database partitions or nodes. This could result in a password being changed only at the database partition to which the user connects to make the change.

DB2_NUM_FAILOVER_NODES

- Operating system: All
- Default: 2, Values: 0 to the number of logical nodes
- Specifies the number of nodes that can be used as failover nodes in a high availability environment. With high availability, if a node fails, then the node can be restarted as a second logical node on a different host. The number used with this variable determines how much memory is reserved for FCM resources for failover nodes.

  For example, host A has two logical nodes: 1 and 2; and host B has two logical nodes: 3 and 4. Assume DB2_NUM_FAILOVER_NODES is set to 2. During DB2START, both host A and host B will reserve enough memory for FCM so that up to four logical nodes could be managed. Then if one host fails, the logical nodes for the failing host could be restarted on the other host.

DB2_PARTITIONEDLOAD_DEFAULT

- Operating system: All supported ESE platforms
- Default: YES, Values: YES or NO
- The DB2_PARTITIONEDLOAD_DEFAULT registry variable lets users change the default behavior of the Load utility in an ESE environment when no ESE-specific Load options are specified. The default value is YES, which specifies that in an ESE environment if you do not specify ESE-specific Load options, loading is attempted on all database partitions on which the target table is defined.

  When the value is NO, loading is attempted only on the database partition to which the Load utility is currently connected.

DB2PORTRANGE

- Operating system: Windows
- Values: nnnn:nnnn
- This value is set to the TCP/IP port range used by FCM so that any additional database partitions created on another machine will also have the same port range.

Related concepts:

- "DB2 registry and environment variables" on page 531

Query compiler variables

DB2_ANTIJOIN

- Operating system: All
- Default=NO in a ESE environment, Default=YES in a non-ESE environment, Values: YES, NO or EXTEND
- For DB2 Enterprise Server Edition: when ″YES″ is specified, the optimizer searches for opportunities to transform "NOT EXISTS" subqueries into anti-joins which can be processed more efficiently by DB2. For non-ESE environments: when ″NO″ is specified, the optimizer limits the opportunities to transform "NOT EXISTS" subqueries into anti-joins.

  In both ESE and NON-ESE environments, when EXTEND is specified, the optimizer searches for opportunities to transform both ″NOT IN″ and ″NOT EXISTS″ subqueries into anti-joins.

DB2_INLIST_TO_NLJN

- Operating system: All
- Default=NO, Values: YES or NO
- In some situations, the SQL and XQuery compiler can rewrite an IN list predicate to a join. For example, the following query:

```
SELECT *
 FROM EMPLOYEE
 WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

  could be written as:

```
SELECT *
 FROM EMPLOYEE, (VALUES 'D11', 'D21', 'E21') AS V(DNO)
 WHERE DEPTNO = V.DNO
```

  This revision might provide better performance if there is an index on DEPTNO. The list of values would be accessed first and joined to EMPLOYEE with a nested loop join using the index to apply the join predicate.

  Sometimes the optimizer does not have accurate information to determine the best join method for the rewritten version of the query. This can occur if the IN list contains parameter markers or host variables which prevent the optimizer from using catalog statistics to determine the selectivity. This registry variable causes the optimizer to favor nested loop joins to join the list of values, using the table that contributes the IN list as the inner table in the join.

Note: When either or both of the DB2 query compiler variables DB2_MINIMIZE_LISTPREFETCH and DB2_INLIST_TO_NLJN, are set to YES, they remain active even if REOPT(ONCE) is specified.

DB2_LIKE_VARCHAR

- Operating system: All
- Default=Y,Y,
- Controls the use of sub-element statistics. These are statistics about the content of data in columns when the data has a structure in the form of a series of sub-fields or sub-elements delimited by blanks. Collection of sub-element statistics is optional and controlled by options in the RUNSTATS command or API.

  This registry variable affects how the optimizer deals with a predicate of the form:

  ```
  COLUMN LIKE '%xxxxxx%'
  ```

  where the xxxxxx is any string of characters.

  The syntax showing how this registry variable is used is:

  ```
  db2set DB2_LIKE_VARCHAR=[Y|N|S|num1] [,Y|N|S|num2]
  ```

  where

  – The term preceding the comma, or the only term to the right of the predicate, means the following but only if the second term is specified as N or the column does not have positive sub-element statistics:
    - S – The optimizer estimates the length of each element in a series of elements concatenated together to form a column based on the length of the string enclosed in the % characters.
    - Y – The default. Use a default value of 1.9 for the algorithm parameter. Use a variable-length sub-element algorithm with the algorithm parameter.
    - N – Use a fixed-length sub-element algorithm.
    - num1 – Use the value of num1 as the algorithm parameter with the variable length sub-element algorithm.

  – The term following the comma means the following, but only for columns that do have positive sub-element statistics:
    - N – Do not use sub-element statistics. The first term takes effect
    - Y – The default. Use a variable-length sub-element algorithm that uses sub-element statistics together with the 1.9 default value for the algorithm parameter in the case of columns with positive sub-element statistics.
    - num2 – Use a variable-length sub-element algorithm that uses sub-element statistics together with the value of num2 as the algorithm parameter in the case of columns with positive sub-element statistics.

DB2_MINIMIZE_LISTPREFETCH

- Operating system: All
- Default=NO, Values: YES or NO
- List prefetch is a special table access method that involves retrieving the qualifying RIDs from the index, sorting them by page number and then prefetching the data pages. Sometimes the optimizer does not have accurate information to determine if list prefetch is a good access

method. This might occur when predicate selectivities contain parameter markers or host variables that prevent the optimizer from using catalog statistics to determine the selectivity.

This registry variable prevents the optimizer from considering list prefetch in such situations.

Note: When either or both of the DB2 query compiler variables DB2_MINIMIZE_LISTPREFETCH and DB2_INLIST_TO_NLJN, are set to YES, they remain active even if REOPT(ONCE) is specified.

DB2_NEW_CORR_SQ_FF

- Operating system: All
- Default=OFF, Values: ON or OFF
- Affects the selectivity value computed by the query optimizer for certain subquery predicates when it is set to "ON". It can be used to improve the accuracy of the selectivity value of equality subquery predicates that use the MIN or MAX aggregate function in the SELECT list of the subquery. For example:

```
SELECT * FROM T WHERE
T.COL = (SELECT MIN(T.COL)
FROM T WHERE ...)
```

DB2_OPT_MAX_TEMP_SIZE

- Operating system: All
- Default: null, Value: amount of space in megabytes that can be used by a query in all temporary table spaces
- Limits the amount of space that queries can use in the temporary table spaces. Setting DB2_OPT_MAX_TEMP_SIZE can cause the optimizer to choose a plan that is more expensive than would otherwise be chosen, but which uses less space in the temporary table spaces. If you set DB2_OPT_MAX_TEMP_SIZE, be sure to balance your need to limit use of temporary table space against the efficiency of the plan your setting causes to be chosen.

  If DB2_WORKLOAD=SAP is set, DB2_OPT_MAX_TEMP_SIZE is automatically set to 10240 (10GB).

  If you run a query that uses temporary table space in excess of the value set for DB2_OPT_MAX_TEMP_SIZE, the query does not fail, but you receive a warning that its performance may be suboptimal, as not all resources may be available.

  The operations considered by the optimizer that are affected by the limit set by DB2_OPT_MAX_TEMP_SIZE are:

  – Explicit sorts for operations such as ORDER BY, DISTINCT, GROUP BY, merge scan joins, and nested loop joins.
  – Explicit temporary tables
  – Implicit temporary tables for hash joins and duplicate merge joins

DB2_PRED_FACTORIZE

- Operating system: All
- Default=NO, Value: YES or NO
- Specifies whether the optimizer searches for opportunities to extract additional predicates from disjuncts. In some circumstances, the additional predicates can alter the estimated cardinality of the intermediate and final result sets. With the following query:

```
SELECT n1.empno,
       n1.lastname
  FROM employee n1,
       employee n2
  WHERE
   ((n1.lastname='SMITH'
  AND n2.lastname='JONES')
  OR (n1.lastname='JONES'
  AND n2.lastname='SMITH'))
```

the optimizer can generate the following additional predicates:

```
SELECT n1.empno,
       n1.lastname
  FROM employee n1,
       employee n2
  WHERE n1.lastname IN
   ('SMITH', 'JONES')
  AND n2.lastname IN
   ('SMITH', 'JONES')
  AND
   ((n1.lastname='SMITH'
  AND n2.lastname='JONES')
  OR (n1.lastname='JONES'
  AND n2.lastname='SMITH'))
```

DB2_REDUCED_OPTIMIZATION

- Operating system: All
- Default=NO, Values: NO, YES, Any integer, DISABLE, NO_SORT_NLJOIN, NO_SORT_MGJOIN
- This registry variable lets you request either reduced optimization features or rigid use of optimization features at the specified optimization level. If you reduce the number of optimization techniques used, you also reduce time and resource use during optimization.

  Note: Although optimization time and resource use might be reduced, the risk of producing a less than optimal data access plan is increased. Use this registry variable only when advised by IBM or one of its partners.

  – If set to NO

    The optimizer does not change its optimization techniques.

  – If set to YES

    If the optimization level is 5 (the default) or lower, the optimizer disables some optimization techniques that might consume significant prepare time and resources but do not usually produce a better access plan.

    If the optimization level is exactly 5, the optimizer scales back or disables some additional techniques, which might further reduce optimization time and resource use, but also further increase the risk of a less than optimal access plan. For optimization levels lower than 5, some of these techniques might not be in effect in any case. If they are, however, they remain in effect.

  – If set to any integer

    The effect is the same as YES, with the following additional behavior for dynamically prepared queries optimized at level 5. If the total number of joins in any query block exceeds the setting, then the optimizer switches to greedy join enumeration instead of disabling additional optimization techniques as described above for level 5

optimization levels. which implies that the query will be optimized at a level similar to optimization level 2.

– If set to DISABLE

  The behavior of the optimizer when unconstrained by this DB2_REDUCED_OPTIMIZATION variable is sometimes to dynamically reduce the optimization for dynamic queries at optimization level 5. This setting disables this behavior and requires the optimizer to perform full level 5 optimization.

– If set to NO_SORT_NLJOIN

  The optimizer does not generate query plans that force sorts for nested loop joins (NLJN). These types of sorts can be useful for improving performance; therefore, be careful when using the NO_SORT_NLJOIN option, as performance can be severely impacted.

– If set to NO_SORT_MGJOIN

  The optimizer does not generate query plans that force sorts for merge scan joins (MSJN). These types of sorts can be useful for improving performance; therefore, be careful when using the NO_SORT_MGJOIN option, as performance can be severely impacted.

  Note that the dynamic optimization reduction at optimization level 5 takes precedence over the behavior described for optimization level of exactly 5 when DB2_REDUCED_OPTIMIZATION is set to YES as well as the behavior described for the integer setting.

DB2_SELECTIVITY

- Operating system: All
- Default=No, Values: Yes or No
- This registry variable controls where the SELECTIVITY clause can be used in search conditions in SQL statements.

  When this registry variable is set to "Yes", the SELECTIVITY clause can be specified for the following predicates:

  – A basic predicate in which at least one expression contains host variables
  – A LIKE predicate in which the MATCH expression, predicate expression, or escape expression contains host variables

DB2_SQLROUTINE_PREPOPTS

- Operating system: All
- Default=empty string, Values:
  – BLOCKING {UNAMBIG | ALL | NO}
  – DATETIME {DEF | USA | EUR | ISO | JIS | LOC}
  – DEGREE {1 | *degree-of-parallelism* | ANY}
  – DYNAMICRULES {BIND | RUN}
  – EXPLAIN {NO | YES | ALL}
  – EXPLSNAP {NO | YES | ALL}
  – FEDERATED {NO | YES}
  – INSERT {DEF | BUF}
  – ISOLATION {CS | RR | UR | RS | NC}
  – QUERYOPT *optimization-level*
  – VALIDATE {RUN | BIND}

- The DB2_SQLROUTINE_PREPOPTS registry variable can be used to customize the precompile and bind options for SQL and XQuery procedures.

Related concepts:

- "DB2 registry and environment variables" on page 531
- "Choosing an optimization class" on page 153
- "Strategies for selecting optimal joins" on page 191

Related reference:

- "Optimization classes" on page 154

Performance variables

DB2_ALLOCATION_SIZE

- Operating system: All
- Default=128 KB, Range: 64 KB–256 MB
- Specifies the size of memory allocations for buffer pools.

  The potential advantage of setting a higher value for this registry variable is fewer allocations will be required to reach a desired amount of memory for a buffer pool.

  The potential cost of setting a higher value for this registry variable is wasted memory if the buffer pool is altered by a non-multiple of the allocation size. For example, if the value for DB2_ALLOCATION_SIZE is 8 MB and a buffer pool is reduced by 4 MB, this 4 MB will be wasted because an entire 8 MB segment cannot be freed.

DB2_APM_PERFORMANCE

- Operating system: All
- Default=OFF, Values: ON , OFF
- Set this variable to ON to enable performance-related changes in the access plan manager (APM) that affect the behavior of the query cache (package cache). These settings are not usually recommended for production systems. They introduce some limitations, such as the possibility of out-of-package cache errors or increased memory use or both.

  Setting DB2_APM_PERFORMANCE to ON also enables the 'No Package Lock' mode. This mode allows the global query cache to operate without the use of package locks, which are internal system locks that protect cached package entries from being removed. The 'No Package Lock' mode might result in somewhat improved performance, but certain database operations are not allowed. These prohibited operations might include: operations that invalidate packages, operations that inoperate packages, and PRECOMPILE, BIND, and REBIND.

DB2ASSUMEUPDATE

- Operating system: All
- Default=OFF, Values: ON, OFF
- When enabled, this variable allows the DB2 database system to assume that all fixed-length columns provided in an UPDATE statement are being changed. This eliminates the need for the DB2 database system to compare the existing column values to the new values to determine if the column is actually changing. Using this registry variable can cause

additional logging and index maintenance when columns are provided for update (for example, in a SET clause) but are not actually being modified.

This registry variable is checked at the time of an update operation.

DB2_AVOID_PREFETCH

- Operating system: All
- Default=OFF, Values: ON or OFF
- Specifies whether prefetch should be used during crash recovery. If DB2_AVOID_PREFETCH=ON, prefetch is not used.

DB2_AWE

- Operating system: Windows 2000
- Default=null, Values: <entry>[;<entry>...] where <entry>=<buffer pool ID>,<number of physical pages>, <number of address windows>
- This variable allows DB2 database systems on 32-bit Windows 2000 platforms to allocate buffer pools that use up to 64 GB of memory. Windows 2000 must be configured to support Address Windowing Extensions (AWE) buffer pools. This includes associating the "lock pages in memory"-right with the user, allocating the physical pages and the address window pages, and setting this registry variable. You need to know the buffer pool ID of the buffer pool that is to be used for AWE support when you set this variable. The buffer pool ID can be seen in the BUFFERPOOLID column of the SYSCAT.BUFFERPOOLS system catalog view.

  Note:

  - Buffer pools that are referenced with this registry variable must already exist in the SYSCAT.SYSBUFFERPOOLS system catalog.
  - Buffer pools that are enabled for AWE will take precedence over buffer pools enabled for block-based I/O. If a buffer pool is configured for both AWE and block-based I/O, AWE will take precedence over block-based I/O.

DB2BPVARS

- Operating system: As specified for each parameter
- Default=path
- Two sets of parameters are available to tune buffer pools. One set of parameters, available only on Windows, specify that buffer pools should use scatter read for specific types of containers. The other set of parameters, available on all platforms, affect prefetching behavior.

  Parameters are specified in an ASCII file, one parameter on each line, in the form `parameter=value`. For example, a file named `bpvars.vars` might contain the following lines:

  ```
  NO_NT_SCATTER = 1
  NUMPREFETCHQUEUES = 2
  ```

  Assuming that `bpvars.vars` is stored in F:\vars\, to set these variables you execute the following command:

  ```
  db2set DB2BPVARS=F:\vars\bpvars.vars
  ```

  Scatter-read parameters

  The scatter-read parameters are recommended for systems with a large amount of sequential prefetching against the respective type of

containers and for which you have already set DB2NTNOCACHE to ON. These parameters, available only on Windows platforms, are NT_SCATTER_DMSFILE, NT_SCATTER_DMSDEVICE, and NT_SCATTER_SMS. Specify the NO_NT_SCATTER parameter to explicitly disallow scatter read for any container. Specific parameters are used to turn scatter read on for all containers of the indicated type. For each of these parameters, the default is zero (or OFF); and the possible values include: zero (or OFF) and 1 (or ON).

Note: You can turn on scatter read only if DB2NTNOCACHE is set to ON to turn Windows file caching off. If DB2NTNOCACHE is set to OFF or not set, a warning message is written to the administration notification log if you attempt to turn on scatter read for any container, and scatter read remains disabled.

Prefetch-adjustment parameters

The prefetch-adjustment parameters are NUMPREFETCHQUEUES and PREFETCHQUEUESIZE. These parameters are available on all platforms and can be used to improve buffer-pool data prefetching. For example, consider sequential prefetching in which the desired PREFETCHSIZE is divided into PREFETCHSIZE/EXTENTSIZE prefetch requests. In this case, requests are placed on prefetch queues from which I/O servers are dispatched to perform asynchronous I/O. By default, the DB2 database manager maintains one queue of size `max( 100 , 2*NUM_IOSERVERS )` for each database partition. In some environments, performance improves with either more queues or queues of a different size, or both. The number of prefetch queues should be at most one half of the number of I/O servers. When you set these parameters, consider other parameters such as PREFETCHSIZE, EXTENTSIZE, NUM_IOSERVERS, and buffer pool size, as well as workload characteristics such as the number of current users.

If you think the default values are too small for your environment, first increase the values only slightly. For example, you might set NUMPREFETCHQUEUES=4 and PREFETCHQUEUESIZE=200. Make changes to these parameters in a controlled manner so that you can monitor and evaluate the effects of the change.

For NUMPREFETCHQUEUES, the default is 1, and the range of values is 1 to NUM_IOSERVERS. If you set NUMPREFETCHQUEUES to less than 1, it is adjusted to 1. If you set it greater than NUM_IOSERVERS, it is adjusted to NUM_IOSERVERS.

For PREFETCHQUEUESIZE, the default value is `max(100,2*NUM_IOSERVERS)`. The range of values is 1 to 32767. If you set PREFETCHQUEUESIZE to less than 1, it is adjusted to the default. If set greater than 32 767, it is adjusted to 32 767.

DB2CHKPTR

- Operating system: All
- Default=OFF, Values: ON or OFF
- Specifies whether or not pointer checking for input is required.

DB2CHKSQLDA

- Operating system: All
- Default=ON, Values: ON or OFF
- Specifies whether or not SQLDA checking for input is required.

DB2_ENABLE_BUFPD

- Operating system: All
- Default=YES, Values: ON or OFF
- Specifies whether or not the DB2 database manager uses intermediate buffering to improve query performance. The buffering may not improve query performance in all environments. Testing should be done to determine individual query performance improvements.

DB2_EVALUNCOMMITTED

- Operating system: All
- Default=OFF, Values: ON, OFF
- When enabled, this variable allows, where possible, table or index access scans to defer or avoid row locking until a data record is known to satisfy predicate evaluation.

  With this variable enabled, predicate evaluation may occur on uncommitted data.

  DB2_EVALUNCOMMITTED is applicable only to statements using either Cursor Stability or Read Stability isolation levels. For index scans, the index must be a type-2 index.

  Furthermore, deleted rows are skipped unconditionally on table scan access while deleted keys are not skipped for type-2 index scans unless the registry variable DB2_SKIPDELETED is also set.

  The activation of the DB2_EVALUNCOMMITTED registry variable is effective on the **db2start** command. The decision as to whether deferred locking is applicable is made at statement compile or bind time.

DB2_EXTENDED_IO_FEATURES

- Operating system: AIX
- Default=OFF, Values: [ON | OFF], [ONLINE_BACKUP_IOPRIORITY=[HIGH | MEDIUM | LOW]]
- Set this variable to ON to enable features that enhance I/O performance. This enhancement includes improving the hit rate of memory caches as well as reducing the latency on high priority I/O. These features are only available on certain combinations of software and hardware configuration; setting this variable to ON for other configurations will be ignored by either the DB2 database management system or by the operating system. The minimum configuration requirements are:
  - Database version: DB2 V9.1
  - RAW device must be used for database containers (container on file systems is not supported)
  - Operating system: AIX 5.3 TL4
  - Storage subsystem: Shark DS8000 supports all the enhanced I/O performance features. Shark DS6000 supports only a subset of the features. Refer to their respective user documentation for prerequisites and setup information.

  If you set DB2_EXTENDED_IO_FEATURES to ON, you can optionally use the ONLINE_BACKUP_IOPRIORITY keyword to set the I/O priority for online backup to HIGH, MEDIUM, or LOW. The default I/O priority settings for HIGH, MEDIUM, and LOW are 3, 8, and 12, respectively; you can use the DB2_IO_PRIORITY_SETTINGS registry variable to change these settings. If you set DB2_EXTENDED_IO_FEATURES to ON, but do not specify the ONLINE_BACKUP_IOPRIORITY keyword, the I/O priority is set to LOW.

DB2_EXTENDED_OPTIMIZATION

- Operating system: All
- Default=OFF, Values: ON or OFF
- Specifies whether or not the query optimizer uses optimization extensions to improve query performance. The extensions may not improve query performance in all environments. Testing should be done to determine individual query performance improvements.

DB2_IO_PRIORITY_SETTING

- Operating system: AIX
- Values: HIGH:#,MEDIUM:#,LOW:#, where # can be 1 to 15
- This variable is used in combination with the DB2_EXTENDED_IO_FEATURES registry variable. This registry variable provides a means to override the default HIGH, MEDIUM, and LOW I/O priority settings for the DB2 database system, which are 3, 8, and 12, respectively. The range of valid I/O priority settings is 1 to 15. This registry variable must be set prior to the start of an instance; any modification requires an instance restart. Note that setting this registry variable alone does not enable the enhanced I/O features, DB2_EXTENDED_IO_FEATURES must be set to enable them. All system requirements for DB2_EXTENDED_IO_FEATURES also apply to this registry variable.

DB2_KEEPTABLELOCK

- Operating system: All
- Default=OFF, Values: ON, OFF
- When enabled, this variable allows the DB2 database system to maintain the table lock when an Uncommitted Read or Cursor Stability isolation level is closed. The table lock that is kept is released at the end of the transaction, just as it would be released for Read Stability and Repeatable Read scans.

  This registry variable is checked at statement compile or bind time.

DB2_LARGE_PAGE_MEM

- Operating system: AIX 5.x 64-bit only, Linux, Windows Server 2003
- Default=NULL, Values: Use * to denote that all applicable memory regions should use large page memory, or a comma-separated list of specific memory regions that should use large page memory. Available regions vary by operating system. On AIX 5.x 64-bit, the following regions can be specified: DB, DBMS, FCM, or PRIVATE. On Linux, the following region can be specified: DB. On Windows Server 2003, the following region can be specified: DB.
- The DB2_LARGE_PAGE_MEM registry variable is used to enable large page support. This registry variable makes the DB2_LGPAGE_BP registry variable unnecessary and replaces it. DB2_LGPAGE_BP was only used to enable large-page memory for the database shared memory region. This can now be enabled by setting DB2_LARGE_PAGE_MEM=DB. Any documentation that mentions enabling large pages with DB2_LGPAGE_BP registry variable can be treated as being synonymous with setting DB2_LARGE_PAGE_MEM=DB.

  Large page usage is primarily intended to provide performance improvements to high performance computing applications. Memory access-intensive applications that use large amounts of virtual memory

may obtain performance improvements by using large pages. To enable the DB2 database system to use large pages, you must first configure the operating system to use large pages.

Enabling large private pages increases DB2 database system memory usage by a significant amount, as each DB2 agent consumes at least 1 large page (16 MB) of physical memory. To enable large pages for agent private memory on 64-bit DB2 for AIX (the DB2_LARGE_PAGE_MEM=PRIVATE setting), the following conditions must be met, in addition to configuring large pages on the operating system:

– The instance owner must possess the CAP_BYPASS_RAC_VMM and CAP_PROPAGATE capabilities.
– The kernel must support interfaces that allow a process to modify its page size at run time.

On 64-bit DB2 for AIX, enabling this variable reduces the size of the shared memory segment backing database memory to the minimum requirement. The default is to create a 64 GB segment: see the database shared memory size (*database_memory*) database configuration parameter for more details. This avoids pinning more shared memory in RAM than is likely to be used.

Setting this variable limits the ability to dynamically increase the overall database shared memory configuration. For example, there is a limit placed on how much the size of the buffer pools can increase.

On 64-bit DB2 for AIX, setting this variable to DB means that self tuning for database shared memory (activated by setting the *database_memory* configuration parameter to AUTOMATIC) cannot be enabled.

On DB2 for AIX 5L™ 64-bit, you can set this variable to FCM. FCM memory resides in its own memory set, so you must add the FCM keyword to the value of the DB2_LARGE_PAGE_MEM registry variable to enable large pages for FCM memory.

On Linux, there is an additional requirement for the availability of the *libcap.so.1* library. This library must be installed for this option to work. If this option is turned on and the library is not on the system, the DB2 database disables the large kernel pages and continues to function as it would without them.

On Linux, to verify that large kernel pages are available, issue the following command:

```
cat /proc/meminfo
```

If large kernel pages are available, the following three lines should appear (with different numbers depending on the amount of memory configured on your server):

```
HugePages_Total:   200
HugePages_Free:    200
Hugepagesize:    16384 kB
```

If you do not see these lines, or if the `HugePages_Total` is 0, you need to configure the operating system or kernel.

On Windows, the amount of large page memory that is available on the system is less than the total available memory. After the system has been running for some time, memory can become fragmented, and the amount of large page memory decreases.

DB2_LGPAGE_BP

- Operating system: AIX 5.x 64-bit only, Linux

- Default=OFF, Values: ON or OFF
- The DB2_LGPAGE_BP registry variable has been deprecated. Please see the DB2_LARGE_PAGE_MEM registry variable.

DB2MAXFSCRSEARCH

- Operating system: All
- Default=5, Values: -1, 1 to 33 554
- Specifies the number of free-space control records to search when adding a record to a table. The default is to search five free-space control records. Modifying this value allows you to balance insert speed with space reuse. Use large values to optimize for space reuse. Use small values to optimize for insert speed. Setting the value to -1 forces the database manager to search all free-space control records.

DB2_MAX_NON_TABLE_LOCKS

- Operating system: All
- Default=YES, Values: See description
- This variable defines the maximum number of NON table locks a transaction can have before it releases all of these locks. NON table locks are table locks that are kept in the hash table and transaction chain even when the transaction has finished using them. Because transactions often access the same table more than once, retaining locks and changing their state to NON can improve performance.

  For best results, the recommended value for this variable is the maximum number of tables expected to be accessed by any connection. If no user-defined value is specified, the default value is as follows: If the locklist size is greater than or equal to

  `SQLP_THRESHOLD_VAL_OF_LRG_LOCKLIST_SZ_FOR_MAX_NON_LOCKS`

  (currently 8000), the default value is

  `SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_LARGE`

  (currently 150). Otherwise, the default value is

  `SQLP_DEFAULT_MAX_NON_TABLE_LOCKS_SMALL`

  (currently 0).

DB2_MDC_ROLLOUT

- Operating system: All
- Default=ON, Values: ON or OFF
- This variable enables a performance enhancement known as “roll out” for deletions from MDC tables.

  Roll out is a faster way of deleting rows in an MDC table, when entire cells (intersections of dimension values) are deleted in a search DELETE statement. The benefits are reduced logging and more efficient processing. A commit operation should be done after such a deletion and before insertions into the table, to allow the emptied blocks to be reused.

DB2MEMDISCLAIM

- Operating system: AIX
- Default=YES, Values: YES or NO
- On AIX, memory used by DB2 database system processes may have some associated paging space. This paging space may remain reserved

even when the associated memory has been freed. Whether or not this is so depends on the AIX system's (tunable) virtual memory management allocation policy. The DB2MEMDISCLAIM registry variable controls whether DB2 agents explicitly request that AIX disassociate the reserved paging space from the freed memory.

A DB2MEMDISCLAIM setting of YES results in smaller paging space requirements, and possibly less disk activity from paging. A DB2MEMDISCLAIM setting of NO will result in larger paging space requirements, and possibly more disk activity from paging. In some situations, such as if paging space is plentiful and real memory is so plentiful that paging never occurs, a setting of NO provides a minor performance improvement.

DB2MEMMAXFREE

- Operating system: All
- Default= 8 388 608 bytes, Values: 0 to 2^{32}-1 bytes
- Specifies the maximum number of bytes of unused private memory that is retained by DB2 database system processes before unused memory is returned to the operating system.

DB2_MEM_TUNING_RANGE

- Operating system: AIX, Windows
- Default=null, Values: a sequence of percentages n, m where n=*minfree* and m=*maxfree*
- The amount of physical memory that the DB2 instance leaves free is important because this dictates how much memory other applications running on the same machine are able to use. When self tuning of database shared memory is enabled, the amount of physical memory left free by a given instance depends on the need for memory by the instance (and its active databases). When an instance is in urgent need of additional memory, it will allocate memory until the free physical memory on the system reaches the percentage specified by *minfree*. When the instance is less in need of memory, it will maintain a larger amount of free physical memory, specified as a percentage by *maxfree*. As a result, it is a requirement that the value set for *minfree* must be less than the value of *maxfree*.

  If this variable is not set, the DB2 database manager will calculate values for *minfree* and *maxfree* based on the amount of memory on the server. It is recommended that this variable *not* be set, unless you are running the self-tuning memory manager (STMM), have DATABASE_MEMORY set to AUTOMATIC, and are experiencing problems related to an insufficient amount of free physical memory.

DB2_MMAP_READ

- Operating system: AIX
- Default=ON, Values: ON or OFF
- This variable is used in conjunction with DB2_MMAP_WRITE to allow the DB2 database system to use mmap as an alternate method of I/O. In most environments, mmap should be used to avoid operating system locks when multiple processes are writing to different sections of the same file.

  When these variables are set to ON, data that is read to and written from the DB2 buffer pools bypasses the AIX memory cache. If you have a relatively small DB2 buffer pool, and you cannot or choose not to

increase the size of this buffer pool, you should take advantage of AIX memory caching by setting DB2_MMAP_READ and DB2_MMAP_WRITE to OFF.

DB2_MMAP_WRITE

- Operating system: AIX
- Default=ON, Values: ON or OFF
- This variable is used in conjunction with DB2_MMAP_READ to allow the DB2 database system to use mmap as an alternate method of I/O. In most environments, mmap should be used to avoid operating system locks when multiple processes are writing to different sections of the same file.

  When these variables are set to ON, data that is read to and written from the DB2 buffer pools bypasses the AIX memory cache. If you have a relatively small DB2 buffer pool, and you cannot or choose not to increase the size of this buffer pool, you should take advantage of AIX memory caching by setting DB2_MMAP_READ and DB2_MMAP_WRITE to OFF.

DB2_NO_FORK_CHECK

- Operating system: UNIX
- Default=OFF, Values: ON, OFF
- When this variable is enabled, the DB2 runtime client minimizes checks to determine if the current process is a result of a fork call. This can improve performance of DB2 applications that do not use the fork() api.

DB2_NO_MPFA_FOR_NEW_DB

- Operating system: All
- Default= not set, Values: YES
- Databases created by the CREATE DATABASE command or the equivalent API have multipage file allocation (MPFA) enabled. Once MPFA is enabled for a database, it cannot be disabled. To create a database with MPFA disabled, set this registry variable to YES and restart the instance before creating the database. When this registry variable is set, any created database will have MPFA disabled.

  To enable MPFA for a database that has MPFA disabled, use the **db2empfa** command.

DB2NTMEMSIZE

- Operating system: Windows
- Default=(varies by memory segment)
- Windows requires that all shared memory segments be reserved at DLL initialization time in order to guarantee matching addresses across processes. DB2NTMEMSIZE permits the user to override the DB2 defaults on Windows if necessary. In most situations, the default values should be sufficient. The memory segments, default sizes, and override options are: 1) Database Kernel: default size is 16777216 (16 MB); override option is DBMS:<number of bytes>. 2) Parallel FCM Buffers: default size is 22020096 (21 MB); override option is FCM:<number of bytes>. 3) Database Admin GUI: default size is 33554432 (32 MB); override option is DBAT:<number of bytes>. 4) Fenced Stored Procedures: default size is 16777216 (16 MB); override option is APLD:<number of bytes>. More than one segment may be overridden

by separating the override options with a semi-colon (;). For example, to limit the database kernel to approximately 256K, and the FCM buffers to approximately 64 MB, use:

```
db2set DB2NTMEMSIZE=DBMS:256000;FCM:64000000
```

DB2NTNOCACHE

- Operating system: Windows
- Default=OFF, Values: ON or OFF
- The DB2NTNOCACHE registry variable specifies whether the DB2 database system opens database files with a NOCACHE option. If DB2NTNOCACHE=ON, file system caching is eliminated. If DB2NTNOCACHE=OFF, the operating system caches DB2 files. This applies to all data except for files that contain long fields or LOBs. Eliminating system caching allows more memory to be available to the database so that the buffer pool or sortheap can be increased.

  In Windows, files are cached when they are opened, which is the default behavior. One MB is reserved from a system pool for every 1 GB in the file. Use this registry variable to override the undocumented 192 MB limit for the cache. When the cache limit is reached, an out-of-resource error is given.

DB2NTPRICLASS

- Operating system: Windows
- Default=null, Value: R, H, (any other value)
- Sets the priority class for the DB2 instance (program DB2SYSCS.EXE). There are three priority classes:
  - NORMAL_PRIORITY_CLASS (the default priority class)
  - REALTIME_PRIORITY_CLASS (set by using "R")
  - HIGH_PRIORITY_CLASS (set by using "H")

  This variable is used in conjunction with individual thread priorities (set using DB2PRIORITIES) to determine the absolute priority of DB2 threads relative to other threads in the system.

  Note: Care should be taken when using this variable. Misuse could adversely affect overall system performance.

  For more information, please refer to the *SetPriorityClass()* API in the Win32 documentation.

DB2NTWORKSET

- Operating system: Windows
- Default=1,1
- Used to modify the minimum and maximum working-set size available to the DB2 database manager. By default, when Windows is not in a paging situation, the working set of a process can grow as large as needed. However, when paging occurs, the maximum working set that a process can have is approximately 1 MB. DB2NTWORKSET allows you to override this default behavior.

  Specify DB2NTWORKSET using the syntax DB2NTWORKSET=`min,max`, where min and max are expressed in megabytes.

DB2_OVERRIDE_BPF

- Operating system: All

- Default=not set, Values: a positive numeric number of pages OR <entry>[;<entry>...] where <entry>=<buffer pool ID>,<number of pages>
- This variable specifies the size of the buffer pool, in pages, to be created at database activation or first connection time. It is useful when memory constraints cause failures occur during database activation or first connection. The memory constraint could arise either in the rare case of a real memory shortage or, because of the attempt by the database manager to allocate a large buffer pool, in the case where there were inaccurately configured buffer pools. For example, when even a minimal buffer pool of 16 pages is not brought up by the database manager, then you can try again after specifying a smaller number of pages using this environment variable. The value given to this variable overrides the current buffer pool size.

  You can also use <entry>[;<entry>...] where <entry>=<buffer pool ID>,<number of pages> to temporarily change the size of all or a subset of the buffer pools so that they can start up.

DB2_PINNED_BP

- Operating system: AIX, HP-UX, Linux
- Default=NO, Values: YES or NO
- This variable is used to specify the database global memory (including buffer pools) associated with the database in the main memory on some AIX operating systems. Keeping this database global memory in the system main memory allows database performance to be more consistent.

  For example, if the buffer pool is swapped out of the system main memory, database performance deteriorates. The reduction of disk I/O by having the buffer pools in system memory improves database performance. If other applications require more of the main memory, allow the database global memory to be swapped out of main memory, depending on the system main memory requirements.

  On Linux, in addition to modifying this registry variable, the library, *libcap.so.1* is also required.

  On 64-bit DB2 for AIX, enabling this variable reduces the size of the shared memory segment by reducing the database memory to the minimum requirement (the default is to create a 64GB segment - see the database_memory configuration parameter for more details). This avoids pinning more shared memory in RAM than is likely to be used.

  Setting this variable limits the ability to dynamically increase the overall database shared memory configuration. For example, there is a limit placed on how much the size of the buffer pools can increase.

  On 64-bit DB2 for AIX, setting this variable to YES means that self tuning for database shared memory (activated by setting the *database_memory* configuration parameter to AUTOMATIC) cannot be enabled.

  For HP-UX in a 64-bit environment, in addition to modifying this registry variable, the DB2 instance group must be given the MLOCK privilege. To do this, a user with root access rights performs the following actions:

  1. Adds the DB2 instance group to the `/etc/privgroup` file. For example, if the DB2 instance group belongs to `db2iadm1` group then the following line must be added to the `/etc/privgroup` file:

     ```
     db2iadm1 MLOCK
     ```

2. Issues the following command:

```
setprivgrp -f /etc/privgroup
```

DB2PRIORITIES

- Operating system: All
- Values setting is platform dependent
- Controls the priorities of DB2 processes and threads.

DB2_RESOURCE_POLICY

- Operating system: AIX 5 or higher, all Linux except zSeries (32–bit), Windows Server 2003 or higher
- Default=Not set, Values: valid path to configuration file
- Defines a resource policy which can be used to limit what operating system resources are used by the DB2 database or it contains rules for assigning specific operating system resources to specific DB2 database objects. For example, on AIX, Linux, or Windows operating systems, this registry variable can be used to limit the set of processors that the DB2 database system uses. The extent of resource control varies depending on the operating system.

  On AIX NUMA enabled machines, a policy can be defined which specifies what resource sets the DB2 database system uses. When resource set binding is used, each individual DB2 process is bound to a particular resource set. This can be beneficial in some performance tuning scenarios.

  You can set the registry variable to indicate the path to a configuration file which defines a policy for binding DB2 processes to operating system resources. The resource policy allows you to specify a set of operating system resources to restrict the DB2 database system. Each DB2 process is bound to a single resource of the set. Resource assignment occurs in a circular round robin fashion.

  Sample configuration files:

```
Example 1: Bind all DB2 processes to either CPU 1 or 3.
  <RESOURCE_POLICY>
  <METHOD>CPU</METHOD>
  <RESOURCE>1</RESOURCE>
  <RESOURCE>3</RESOURCE>
  </RESOURCE_POLICY>

Example 2: Bind DB2 processes to one of the following resource sets:
  sys/node.03.00000, sys/node.03.00001, sys/node.03.00002,
sys/node.03.00003
  <RESOURCE_POLICY>
  <METHOD>RSET</METHOD>
  <RESOURCE>sys/node.03.00000</RESOURCE>
  <RESOURCE>sys/node.03.00001</RESOURCE>
  <RESOURCE>sys/node.03.00002</RESOURCE>
  <RESOURCE>sys/node.03.00003</RESOURCE>
  </RESOURCE_POLICY>
```

  Note: Use of the RSET method requires CAP_NUMA_ATTACH capability and is not supported on Linux.

  The configuration file specified by the DB2_RESOURCE_POLICY registry variable accepts a SCHEDULING_POLICY element. You can use the SCHEDULING_POLICY element on some platforms to select

  – The operating system scheduling policy used by the DB2 server

You can set an operating system scheduling policy for DB2 on AIX, and for DB2 on Windows using the DB2NTPRICLASS registry variable.

– The operating system priorities used by individual DB2 server agents

Alternatively, you can use the registry variables DB2PRIORITIES and DB2NTPRICLASS to control the operating system scheduling policy and set DB2 agent priorities. However, the specification of a SCHEDULING_POLICY element in the resource policy configuration file provides a single place to specify both the scheduling policy and the associated agent priorities.

```
Example 1: Selection of the AIX SCHED_FIFO2 scheduling
policy with a priority boost for the db2 log writer and reader processes.
  <RESOURCE_POLICY>
   <SCHEDULING_POLICY>
      <POLICY_TYPE>SCHED_FIFO2</POLICY_TYPE>
      <PRIORITY_VALUE>60</PRIORITY_VALUE>

      <EDU_PRIORITY>
         <EDU_NAME>db2loggr</EDU_NAME>
         <PRIORITY_VALUE>56</PRIORITY_VALUE>
      </EDU_PRIORITY>

      <EDU_PRIORITY>
         <EDU_NAME>db2loggw</EDU_NAME>
         <PRIORITY_VALUE>56</PRIORITY_VALUE>
      </EDU_PRIORITY>
   </SCHEDULING_POLICY>
</RESOURCE_POLICY
```

```
Example 2: Replacement for DB2NTPRICLASS=H on Windows.
<RESOURCE_POLICY>
   <SCHEDULING_POLICY>
      <POLICY_TYPE>HIGH_PRIORITY_CLASS</POLICY_TYPE>
   </SCHEDULING_POLICY>
</RESOURCE_POLICY
```

DB2_SKIPDELETED

- Operating system: All
- Default=OFF, Values: ON, OFF
- When enabled, this variable allows statements using either Cursor Stability or Read Stability isolation levels to unconditionally skip deleted keys during index access and deleted rows during table access. With DB2_EVALUNCOMMITTED enabled, deleted rows are automatically skipped, but uncommitted pseudo-deleted keys in type-2 indexes are not skipped unless DB2_SKIPDELETED is also enabled.

  This registry variable does not impact the behavior of cursors on the DB2 catalog tables.

  This registry variable is activated with the db2start command.

DB2_SKIPINSERTED

- Operating system: All
- Default=OFF, Values: ON, OFF
- When the DB2_SKIPINSERTED registry variable is enabled, it allows statements using either Cursor Stability or Read Stability isolation levels to skip uncommitted inserted rows as if they had not been inserted. This registry variable does not impact the behavior of cursors on the DB2 catalog tables. This registry variable is activated at database startup, while the decision to skip uncommitted inserted rows is made at statement compile or bind time.

DB2_SMS_TRUNC_TMPTABLE_THRESH

- Operating system: All
- Default=0, Values: -1, 0-n, where n=the number of extents per temporary table in the SMS tablespace container that are to be maintained

- This variable specifies a minimum file size threshold at which the file representing a temporary table is maintained in SMS table spaces.

  By default, this variable is set to 0, which means no special threshold handling is done. Instead, once a temporary table is no longer needed, that file is truncated to 0 extent.

  When the value of this variable is greater than 0, a larger file is maintained. This reduces some of the system overhead involved in dropping and recreating the file each time a temporary table is used.

  If this variable is set to -1, the file is not truncated and the file is allowed to grow indefinitely, restricted only by system resources.

DB2_SORT_AFTER_TQ

- Operating system: All
- Default=NO, Values: YES or NO
- Specifies how the optimizer works with directed table queues in a partitioned database environment when the receiving end requires the data to be sorted and the number of receiving nodes is equal to the number of sending nodes.

  When DB2_SORT_AFTER_TQ= `NO`, the optimizer tends to sort at the sending end and merge the rows at the receiving end.

  When DB2_SORT_AFTER_TQ= `YES`, the optimizer tends to transmit the rows unsorted, not merge at the receiving end, and sort the rows at the receiving end after receiving all the rows.

DB2_SELUDI_COMM_BUFFER

- Operating system: All
- Default=OFF, Values=ON, OFF
- This variable is used during the processing of blocking cursors over SELECT from UPDATE, INSERT, or DELETE (UDI) queries. When enabled, this registry variable prevents the result of a query from being stored in a temporary table. Instead, during the OPEN processing of a blocking cursor for a SELECT from UDI query, the DB2 database system attempts to buffer the entire result of the query directly into the communications buffer memory area.

  Note: If the communications buffer space is not large enough to hold the entire result of query, an SQLCODE -906 error is issued, and the transaction is rolled back. See the *aslheapsz* and *rqrioblk* database manager configuration parameters for information on adjusting the size of the communication buffer memory area for local and remote applications respectively.

  This registry variable is not supported in partitioned database environments or when intra-partition parallelism is enabled.

DB2_TRUSTED_BINDIN

- Operating system: All
- Default=OFF, Values=OFF, ON, CHECK
- When DB2_TRUSTED_BINDIN is enabled, it speeds up the execution of query statements containing host variables within an embedded unfenced stored procedure.

  When this variable is enabled, there is no conversion from the external SQLDA format to an internal DB2 format during the binding of SQL and

XQuery statements contained within an embedded unfenced stored procedure. This will speed up the processing of the embedded SQL and XQuery statements.

The following datatypes are not supported in embedded unfenced stored procedures when this variable is enabled:

- SQL_TYP_DATE
- SQL_TYP_TIME
- SQL_TYP_STAMP
- SQL_TYP_DATALINK
- SQL_TYP_CGSTR
- SQL_TYP_BLOB
- SQL_TYP_CLOB
- SQL_TYP_DBCLOB
- SQL_TYP_CSTR
- SQL_TYP_LSTR
- SQL_TYP_BLOB_LOCATOR
- SQL_TYP_CLOB_LOCATOR
- SQL_TYP_DCLOB_LOCATOR
- SQL_TYP_BLOB_FILE
- SQL_TYP_CLOB_FILE
- SQL_TYP_DCLOB_FILE
- SQL_TYP_BLOB_FILE_OBSOLETE
- SQL_TYP_CLOB_FILE_OBSOLETE
- SQL_TYP_DCLOB_FILE_OBSOLETE

If these datatypes are encountered, an SQLCODE −804, SQLSTATE 07002 error is returned.

Note: The data type and length of the input host variable must match the internal data type and length of the corresponding element exactly. For host variables, this requirement will always be met. However, for parameter markers, care must be taken to ensure that matching data types are used. The CHECK option can be used to ensure that the data types and lengths match for all input host variables, but this option negates most of the performance improvements.

DB2_USE_ALTERNATE_PAGE_CLEANING

- Operating system: All
- Default=not set, Values: ON, OFF
- This variable specifies whether a DB2 database uses the alternate method of page cleaning algorithms or the default method of page cleaning. When this variable is set to ″ON,″ the DB2 system writes changed pages to disk, keeping ahead of LSN_GAP and proactively finding victims. Doing this allows the page cleaners to better utilize available disk I/O bandwidth. When this variable is set to ″ON,″ the chngpgs_thresh database configuration parameter is no longer relevant because it does not control page cleaner activity.

Related concepts:

- “DB2 registry and environment variables” on page 531

Data links variables

Table 77. Data Links Variables

Variable Name	Operating System	Values
Description		
DLFM_BACKUP_DIR_NAME	AIX, Windows, Solaris	Default: null Values: any valid path
Specifies the path of the directory to which archived files are backed up when the DLFM_BACKUP_TARGET is set to LOCAL.		
DLFM_BACKUP_TARGET	AIX, Windows, Solaris	Default: null Values: LOCAL, TSM, XBSA
Specifies the backup medium to use. If this variable is set to LOCAL, the DLFM_BACKUP_DIR_NAME variable must be set. If this variable is set to XBSA, you must set the DLFM_BACKUP_TARGET_LIBRARY variable. If you change the setting of this registry variable from one target to another, the archived files are not moved. Only new backups are placed in the new location. Previously archived files are not moved.		
DLFM_BACKUP_TARGET_LIBRARY	AIX, Windows, Solaris	Default: null Values: any valid path to the DLL or shared library name
Specifies the fully qualified path to the XBSA-compliant archive server DLL or shared library. This library is loaded using the *libdfmxbsa.a* library. This variable must be set if the DLFM_BACKUP_TARGET is set to XBSA. It does not apply if the DLFM_BACKUP_TARGET variable is set to another value.		
DLFM_GC_MODE	AIX, Windows, Solaris	Default: PASSIVE Values: SLEEP, PASSIVE, or ACTIVE
Specifies the control of garbage file collection on the Data Links server. When set to SLEEP, no garbage collection occurs. When set to PASSIVE, garbage collection runs only if no other transactions are running. When set to ACTIVE, garbage collection runs even if other transactions are running.		
DLFM_INSTALL_PATH	AIX, Windows, Solaris	Default On AIX and Solaris: `/home/<instance>/sqllib/bin` where <instance> is the Data Links Manager instance ID On Windows: `%DB2PATH%\bin` (if %DB2PATH% is set) or `c:\sqllib\bin` (if %DB2PATH% is not set) Range: any valid path
Specifies the path where the Data Links executables are installed.		
DLFM_PORT	AIX, Windows, Solaris	Default: 50100 Values: any valid port number
Specifies the port number used to communicate with the Data Links server running the DB2 Data Links Manager.		
DLFM_TSM_MGMTCLASS	AIX, Windows, Solaris	Default: the default TSM management class Values: any valid TSM management class

Table 77. Data Links Variables (continued)

Variable Name	Operating System	Values
Description		
Specifies which TSM management class to use to archive and retrieve linked files. If no value is set for this variable, the default TSM management class is used.		
DLFM_START_ASNCOPYD	AIX, Windows, Solaris	Default: NO Values: YES, NO
Specifies whether the Data Links Manager Replication Daemon (DLFM_ASNCOPYD) is started whenever DLFM is started. The Data Links Manager Replication Daemon must be started when using DB2 Replication to copy DATALINK files to or from a Data Links Manager server. When this variable is set to YES, the DLFM_ASNCOPYD_PORT variable must be set.		
DLFM_ASNCOPYD_PORT	AIX, Windows, Solaris	Default: null Values: any valid port number
Specifies the TCP/IP port number on which the Data Links Manager Replication Daemon (DLFM_ASNCOPYD) will listen for file replication requests. This variable must be specified when the DLFM_START_ASNCOPYD variable is set to YES.		
DLFM_NUM_ARCHIVE_SUBSYSTEMS	AIX, Windows, Solaris	Default: 2 Values: any number greater than or equal to 1
Specifies the number of DLFM Copy Daemon processes to run under a given DLFM server. The larger the number of copy processes, the greater the throughput on backing up linked files. However, this value should correspond to the number of I/O channels available for copying linked files to the designated archive area. If the value is too large, the amount of system resources consumed can reduce the benefits of I/O parallelism.		
DLFM_AUTOSTART	AIX, Solaris	Default: NO Values: YES, NO
Specifies whether the DLFM server is automatically started whenever the operating system reboots. This variable is checked by the `dlfsmount` script, invoked from the `/etc/inittab` file during boot processing.		

Related concepts:

- "DB2 registry and environment variables" on page 531

Miscellaneous variables

DB2ADMINSERVER

- Operating system: Windows and UNIX
- Default=null
- Specifies the DB2 Administration Server.

DB2CLIINIPATH

- Operating system: All
- Default=null
- Used to override the default path of the DB2 CLI/ODBC configuration file (db2cli.ini) and specify a different location on the client. The value specified must be a valid path on the client system.

DB2_COMMIT_ON_EXIT

- Operating system: UNIX
- Default=OFF, Values: OFF/NO/0 or ON/YES/1

- On UNIX platforms, prior to version 8, DB2 committed any remaining in-flight transactions on successful application exit. In version 8, the behavior was changed so that in-flight transactions were rolled back on exit. This registry variable allows users with applications which depend on the earlier behavior to continue to enable it in Version 8.

  Note that this registry variable will be deprecated in version 10, and the commit-on-exit behavior will no longer be supported. Users should determine whether any of their applications developed prior to version 8 continue to depend on this functionality, and add the appropriate explicit COMMIT statements to the application as required. If the registry variable is turned on, care should be taken not to implement new applications which fail to explicitly COMMIT before exit.

  Most users should leave this registry variable at the default setting.

DB2_CREATE_DB_ON_PATH

- Operating system: Windows
- Default=null, Values: YES, or NO
- Set this registry variable to YES to enable support for the use of a path (as well as a drive) as a database path. The setting of DB2_CREATE_DB_ON_PATH is checked when a database is created, when the database manager configuration parameter DFTDBPATH is set, and when a database is restored. The fully qualified database path can be up to 215 characters in length.

  If DB2_CREATE_DB_ON_PATH is not set (or is set to NO) and you specify a path for the database path when creating or restoring a database, error SQL1052N is returned.

  If DB2_CREATE_DB_ON_PATH is not set (or is set to NO) and you update the DFTDBPATH database manager configuration parameter, error SQL5136N is returned.

  CAUTION:
  If path support is used to create new databases, applications written prior to Version 9.1 using the db2DbDirGetNextEntry() API or an older version of it, might not work correctly. Please refer to http://www.ibm.com/software/data/db2/udb/support/ for details on various scenarios and the proper course of action.

DB2DEFPREP

- Operating system: All
- Default=NO, Values: ALL, YES, or NO
- Simulates the runtime behavior of the DEFERRED_PREPARE precompile option for applications that were precompiled before this option was available. For example, if a DB2 v2.1.1 or earlier application were run in a DB2 v2.1.2 or later environment, DB2DEFPREP could be used to indicate the desired 'deferred prepare' behavior.

DB2_DISABLE_FLUSH_LOG

- Operating system: All
- Default=OFF, Values: ON or OFF
- Specifies whether to disable closing the active log file when the on-line backup is completed.

  When an on-line backup completes, the last active log file is truncated, closed, and made available to be archived. This ensures that your on-line backup has a complete set of archived logs available for recovery. You might want to disable closing the last active log file if you are concerned

that you are wasting portions of the Log Sequence Number (LSN) space. Each time an active log file is truncated, the LSN is incremented by an amount proportional to the space truncated. If you perform a large number of on-line backups each day, you might disable closing the last active log file.

You might also want to disable closing the last active log file if you find you are receiving log full messages a short time after the completion of the on-line backup. When a log file is truncated, the reserved active log space is incremented by the amount proportional to the size of the truncated log. The active log space is freed once the truncated log file is reclaimed. The reclamation occurs a short time after the log file becomes inactive. During the short interval between these two events, you may receive log full messages.

During any backup which includes logs, this registry variable is ignored, since the active log file must be truncated and closed in order for the backup to include the logs.

DB2_DISCONNECT_ON_INTERRUPT

- Operating system: All
- Default= NO, Values: YES, TRUE, 1 or NO, FALSE, 0
- When set to YES (TRUE or 1), this variable specifies that the connection to a Version 8 DB2 UDB z/OS server should be broken immediately when an interrupt occurs. You can use this variable in the following configurations:
  - If you are running a Version 8 DB2 client with a Version 8 DB2 UDB z/OS server, set DB2_DISCONNECT_ON_INTERRUPT to YES on the client.
  - If you are running a Version 8 DB2 client through a Version 8 DB2 Connect gateway to a Version 8 DB2 UDB z/OS server, set DB2_DISCONNECT_ON_INTERRUPT to YES on the gateway.

DB2_DISPATCHER_PEEKTIMEOUT

- Operating system: All
- Default: 1, Values: 0 to 32767 seconds; 0 denotes that timeout is immediate
- DB2_DISPATCHER_PEEKTIMEOUT allows you to adjust the time, in seconds, that a dispatcher waits for a client's connection request before handing the client off to an agent. In most cases, you should not need to adjust this registry variable. This registry variable only affects instances that have DB2 Connect connection concentrator enabled.

  This registry variable and the DB2_SERVER_CONTIMEOUT registry variable both configure the handling of a new client during connect time. If there are many slow clients connecting to an instance, the dispatcher may be held up for up to 1 second to timeout each client, causing the dispatcher to become a bottle neck, if many clients are connecting simultaneously. If an instance with multiple active databases is experiencing very slow connection times, DB2_DISPATCHER_PEEKTIMEOUT may be lowered to 0. Lowering DB2_DISPATCHER_PEEKTIMEOUT causes the dispatcher to only look into the client's connect request if it is already there; the dispatcher will not wait for the connect request to arrive. If an invalid value is set, the default value is used. This registry variable is not dynamic.

DB2_DJ_COMM

- Operating system: All
- Default=null, Values include: libdb2drda.a, libdb2net8.a, libdb2informix.a db2drda.dll, db2net8.dll, db2informix.a, and so on.
- Specifies the wrapper libraries that are loaded when the database manager is started. Specifying this variable reduces the run-time cost of loading frequently used wrappers. Other values for other operating systems are supported (the .dll extension is for the Windows operating system; the .a extension is for the AIX operating system). Library names vary by protocol and operating system.

  This variable is ignored unless the database manager parameter FEDERATED is set to YES.

DB2_DJ_INI

- Operating system: All
- Default:
  - UNIX: db2_instance_directory/cfg/db2dj.ini
  - Windows: db2_install_directory\cfg\db2dj.ini
- Specifies the absolute path name of the federation configuration file, for example: `db2set DB2_DJ_INI=$HOME/sqllib/cfg/my_db2dj.ini` This file contains the settings for data source environment variables. These environment variables are used by the Informix® wrapper and by the wrappers provided by WebSphere Federation Server.

  Here is a sample federation configuration file:

```
INFORMIXDIR=/informix/client_sdk
INFORMIXSERVER=inf93
ORACLE_HOME=/usr/oracle9i
SYBASE=/sybase/V12
SYBASE_OCS=OCS-12_5
```

  The following restrictions apply to the db2dj.ini file:
  - Entries must follow the format *evname=value* where `evname` is the name of the environment variable and `value` is its value.
  - The environment variable name has a maximum length of 255 bytes.
  - The environment variable value has a maximum length of 765 bytes.

  This variable is ignored unless the database manager parameter FEDERATED is set to YES.

DB2DMNBCKCTLR

- Operating system: Windows
- Default=null, Values: ? or a domain name
- If you know the name of the domain for which the DB2 server is the backup domain controller, set DB2DMNBCKCTLR=`DOMAIN_NAME`. The DOMAIN_NAME must be in upper case. To have DB2 determine the domain for which the local machine is a backup domain controller, set DB2DMNBCKCTLR=?. If the DB2DMNBCKCTLR profile variable is not set or is set to blank, DB2 performs authentication at the primary domain controller.

  Note: DB2 does not use an existing backup domain controller by default because a backup domain controller can get out of synchronization with the primary domain controller, causing a security exposure. Getting out of synchronization can occur when the primary domain controller's security database is updated but

the changes are not propagated to a backup domain controller. This could occur if there are network latencies or if the computer browser service is not operational.

DB2_DOCHOST

- Operating system: All
- Default: http://publib.boulder.ibm.com/infocenter/db2help/ (http://*hostname* where *hostname*= valid host name or IP address)
- Specifies the host name on which the DB2 Information Center is installed. This variable can be automatically set during the installation of the DB2 Information Center if the automatic configuration option is selected in the DB2 Setup wizard.

DB2_DOCPORT

- Operating system: All
- Default: NULL, Values: any valid port number
- Specifies the port number through which the DB2 help system serves the DB2 documentation. This variable can be automatically set during the installation of the DB2 Information Center if the automatic configuration option is selected in the DB2 Setup wizard.

DB2_ENABLE_AUTOCONFIG_DEFAULT

- Operating system: All
- Default: null, Values: YES or NO
- This variable controls whether the Configuration Advisor is run automatically at database creation. If DB2_ENABLE_AUTOCONFIG_DEFAULT is not set (null), the effect is the same as if the variable was set to YES and the Configuration Advisor is run at database creation. You do not need to restart the instance after you set this variable. If you execute the AUTOCONFIGURE command or run CREATE DB AUTOCONFIGURE, these commands override the setting of DB2_ENABLE_AUTOCONFIG_DEFAULT.

DB2_ENABLE_LDAP

- Operating system: All
- Default=NO, Values: YES or NO
- Specifies whether or not the Lightweight Directory Access Protocol (LDAP) is used. LDAP is an access method to directory services.

DB2_EXTSECURITY

- Operating system: Windows
- Default=ON, Values: ON or OFF
- Prevents unauthorized access to DB2 by locking DB2 system files. To avoid potential problems, this registry variable should not be turned off.

DB2_FALLBACK

- Operating system: Windows
- Default=OFF, Values: ON or OFF
- This variable allows you to force all database connections off during the fallback processing. It is used in conjunction with the failover support in the Windows environment with Microsoft Cluster Server (MSCS). If DB2_FALLBACK is not set or is set to OFF, and a database connection exists during the fall back, the DB2 resource cannot be brought offline. This will mean the fallback processing will fail.

DB2_FMP_COMM_HEAPSZ

- Operating system: Windows, UNIX
- Default: 20MB, or enough space to run 10 fenced routines (whichever is larger). On AIX, the default is 256MB
- This variable specifies, in 4 KB pages, the size of the pool used for fenced routine invocations, such as stored procedure or user-defined function calls. The space used by each fenced routine is twice the value of the aslheapsz configuration parameter.

  If you are running a large number of fenced routines on your system, you may need to increase the value of this variable. If you are running a very small number of fenced routines, you can reduce it.

  Setting this value to 0 means that no set is created, and as a result no fenced routines can be invoked. It also means that the health monitor and the automatic database maintenance functionality (such as automatic backups, statistics collection, and REORG) will be disabled since this functionality relies on the fenced routine infrastructure.

DB2_GRP_LOOKUP

- Operating system: Windows
- Default=null, Values: LOCAL, DOMAIN
- This variable specifies which Windows security mechanism is used to enumerate the groups to which a user belongs.

DB2_HADR_BUF_SIZE

- Operating system: All
- Default=2*LOGBUFSZ
- This variable specifies the standby log receiving buffer size in unit of log pages. If not set, DB2 will use two times the primary LOGBUFSZ configuration parameter value for the standby receiving buffer size. This variable should be set in the standby instance. It is ignored by the primary database.

  If HADR synchronization mode (the HADR_SYNCMODE database configuration parameter) is set to ASYNC, during peer state, a slow standby may cause the send operation on the primary to stall and therefore block transaction processing on the primary. A larger than default log-receiving buffer can be configured on a standby database to allow it to hold more unprocessed log data. This may allow for brief periods where the primary generates log data faster than the standby can consume it, without blocking transaction processing at the primary.

DB2LDAP_BASEDN

- Operating system: All
- Default=null, Values: Any valid base distinguished domain name.
- When this is set, the LDAP objects for DB2 will be stored in the LDAP directory under

  ```
  CN=System
  CN=IBM
  CN=DB2
  ```

  under the base distinguished name specified.

  When this is set for the Microsoft Active Directory Server, ensure that CN=DB2, CN=IBM and CN=System are defined under this distinguished name.

DB2LDAPCACHE

- Operating system: All
- Default=YES, Values: YES or NO
- Specifies that the LDAP cache is to be enabled. This cache is used to catalog the database, node, and DCS directories on the local machine.

  To ensure that you have the latest entries in the cache, do the following:

  ```
  REFRESH LDAP DB DIR
  REFRESH LDAP NODE DIR
  ```

  These commands update and remove incorrect entries from the database directory and the node directory.

DB2LDAP_CLIENT_PROVIDER

- Operating system: Windows
- Default=null (Microsoft, if available, is used; otherwise IBM is used.) Values: IBM or Microsoft
- When running in a Windows environment, DB2 supports using either Microsoft LDAP clients or IBM LDAP clients to access the LDAP directory. This registry variable is used to explicitly select the LDAP client to be used by DB2.

  Note: To display the current value of this registry variable, use the `db2set` command:

  ```
  db2set DB2LDAP_CLIENT_PROVIDER
  ```

DB2LDAPHOST

- Operating system: All
- Default=null, Values: Any valid hostname
- Specifies the hostname of the location for the LDAP directory.

DB2LDAP_KEEP_CONNECTION

- Operating system: All
- Default= YES, Values: YES, NO
- Specifies whether DB2 caches its internal LDAP connection handles. When this variable is set to NO, DB2 will not cache its LDAP connection handles to the directory server. This will likely result in a negative performance impact, but it might be desirable to set DB2LDAP_KEEP_CONNECTION to NO if the number of simultaneously active LDAP client connections to the directory server needs to be minimized.

  To maximize performance, this variable is set to YES by default.

  The DB2LDAP_KEEP_CONNECTION registry variable is only implemented as a global level profile registry variable in LDAP, so you must set it by specifying the -gl option with the **db2set** command as follows:

  ```
  db2set -gl DB2LDAP_KEEP_CONNECTION=NO
  ```

DB2LDAP_SEARCH_SCOPE

- Operating system: All
- Default= DOMAIN, Values: LOCAL, DOMAIN, GLOBAL
- Specifies the search scope for information found in database partitions or domains in the Lightweight Directory Access Protocol (LDAP). “LOCAL” disables searching in the LDAP directory. “DOMAIN” only

searches in LDAP for the current directory partition. "GLOBAL" searches in LDAP in all directory partitions until the object is found.

DB2_LOAD_COPY_NO_OVERRIDE

- Operating system: All
- Default: NONRECOVERABLE, Values: COPY YES, NONRECOVERABLE
- This variable will convert any LOAD COPY NO to either LOAD COPY YES or NONRECOVERABLE, depending on the value of the variable. This variable is applicable to HADR primary databases as well as to standard (non-HADR) databases; it is ignored on an HADR standby database. On an HADR primary database, if this variable is not set, LOAD COPY NO is converted to LOAD NONRECOVERABLE. The value of this variable either specifies a nonrecoverable load or the copy destination, using the same syntax as a COPY YES clause.

DB2LOADREC

- Operating system: All
- Default=null
- Used to override the location of the load copy during roll forward. If the user has changed the physical location of the load copy, DB2LOADREC must be set before issuing the roll forward.

DB2LOCK_TO_RB

- Operating system: All
- Default=null, Values: STATEMENT
- Specifies whether lock timeouts cause the entire transaction to be rolled back, or only the current statement. If DB2LOCK_TO_RB is set to `STATEMENT`, locked timeouts cause only the current statement to be rolled back. Any other setting results in transaction rollback.

DB2_MAP_XML_AS_CLOB_FOR_DLC

- Operating system: All
- Default= NO, Values: YES or NO
- The DB2_MAP_XML_AS_CLOB_FOR_DLC registry variable provides the ability to override the default DESCRIBE and FETCH behavior of XML values for clients (or DRDA Application Requestors) that do not support XML as a data type. (These are generally referred to as *down-level* clients.)

  The default value is NO, which specifies that for down-level clients a DESCRIBE of XML values will return BLOB(2GB), and a FETCH of XML values will result in an implicit XML serialization to BLOB that includes an XML declaration indicating an encoding of UTF-8.

  When the value is YES, a DESCRIBE of XML values will return CLOB(2GB), and a FETCH of XML values will result in an implicit XML serialization to CLOB that does not contain an XML declaration.

DB2_MAX_LOB_BLOCK_SIZE

- Operating system: All
- Default=0 (no limit), Values: 0 to 21487483647
- Sets the maximum amount of LOB or XML data to be returned in a block. This is not a hard maximum; if this maximum is reached on the server during data retrieval, the server finishes writing out the current row before generating a reply for the command, such as FETCH, to the client.

DB2_NEWLOGPATH2

- Operating system: UNIX
- Default=0, Values: 0 or 1
- This parameter allows you to specify whether a secondary path should be used to implement dual logging. The secondary path name is generated by appending a "2" to the current value of the *logpath* database configuration parameter.

DB2NOEXITLIST

- Operating system: All
- Default=OFF, Values: ON or OFF
- If defined, this variable indicates to DB2 not to install an exit list handler in applications and not to perform a COMMIT. Normally, DB2 installs a process exit list handler in applications and the exit list handler performs a COMMIT operation if the application ends normally.

  For applications that dynamically load the DB2 library and unload it before the application terminates, the invocation of the exit list handler fails because the handler routine is no longer loaded in the application. If your application operates in this way, you should set the DB2NOEXITLIST variable and ensure your application explicitly invokes all required COMMITs.

DB2_NUM_CKPW_DAEMONS

- Operating system: UNIX
- Default=3, Values: 0 to 100
- You can use the DB2_NUM_CKPW_DAEMONS registry variable to start a configurable number of check password daemons. The daemons are created during db2start, run as root, and they handle check password requests. Increasing the setting for DB2_NUM_CKPW_DAEMONS can decrease the time required to establish a database connection, but this is only effective in scenarios where many connections are being made simultaneously and where authentication is expensive (for example, accessing remote password files over Network Information Service (NIS) or NIS+).

  Setting DB2_NUM_CKPW_DAEMONS to zero disables the check password daemons and causes the DB2 database manager to start a separate program to check a password every time it is needed (version 7 behavior).

DB2REMOTEPREG

- Operating system: Windows
- Default=null, Values: Any valid Windows machine name
- Specifies the remote machine name that contains the Win32 registry list of DB2 instance profiles and DB2 instances. The value for DB2REMOTEPREG should only be set once after DB2 is installed, and should not be modified. Use this variable with extreme caution.

DB2_RESOLVE_CALL_CONFLICT

- Operating system: AIX, HP-UX, Solaris, Linux, Windows
- Default=YES, Values: YES or NO
- Eliminates SQLCODE -746 errors in the context of triggers. When issuing a CALL statement in a trigger, an SQLCODE SQL0746 may be issued at runtime. The SQL0746 error prevents procedures called by a trigger from accessing tables that have been previously modified within the context

of the invoking statement. With this variable set, the DB2 database manager enforces that all modifications to tables are completed in compliance with the SQL Standard rules for triggers before executing the CALL statement.

You must stop the instance before you reset DB2_RESOLVE_CALL_CONFLICT and then restart it. Then rebind any packages which cause invocation of triggers. To rebind SQL Procedures use: CALL SYSPROC.REBIND_ROUTINE_PACKAGE ('P','*procedureschema.procedurename*','CONSERVATIVE');

You need to be aware that DB2_RESOLVE_CALL_CONFLICT can have a performance impact. Setting DB2_RESOLVE_CALL_CONFLICT to YES causes the DB2 database manager to resolve all potential read and write conflicts through the injection of temporary tables, as needed. Typically, the impact is small because at most one temporary table is injected. This has a small effect in an OLTP environment because only one (or a small number of) rows are being modified by the triggering statement. Typically, when following the general recommendation to use SMS (system managed storage) for temporary table spaces, the performance impact from setting DB2_RESOLVE_CALL_CONFLICT is expected to be low.

DB2ROUTINE_DEBUG

- Operating system: AIX and Windows
- Default=OFF, Values: ON, OFF
- Specifies whether to enable the debug capability for Java stored procedures. If you are not debugging Java stored procedures, use the default, OFF. There is a performance impact to enable debugging.

DB2SATELLITEID

- Operating system: All
- Default=null, Values: a valid satellite ID declared in the Satellite Control Database
- Specifies the satellite ID that is passed to the satellite control server when a satellite synchronizes. If a value is not specified for this variable, the logon ID is used as the satellite ID.

DB2_SERVER_CONTIMEOUT

- Operating system: All
- Default=180, Values: 1 to 32767 seconds
- This registry variable and the DB2_DISPATCHER_PEEKTIMEOUT registry variable both configure the handling of a new client during connect time. DB2_SERVER_CONTIMEOUT allows you to adjust the time, in seconds, that an agent waits for a client's connection request before terminating the connection. In most cases, you should not need to adjust this registry variable, but if DB2 clients are constantly being timed out by the server at connect time, you can set a higher value for DB2_SERVER_CONTIMEOUT to extend the timeout period. If an invalid value is set, the default value is used. This registry variable is not dynamic.

DB2SORT

- Operating system: All, server only
- Default=null

- Specifies the location of a library to be loaded at runtime by the LOAD utility. The library contains the entry point for functions used in sorting indexing data. Use DB2SORT to exploit vendor-supplied sorting products for use with the LOAD utility in generating table indexes. The path supplied must be relative to the database server.

DB2_TRUNCATE_REUSESTORAGE

- Operating system: All
- Default=null (not set), Values: IMPORT, import
- You can use this variable to resolve lock contention between the IMPORT with REPLACE command and the BACKUP ... ONLINE command. In some situations, online backup and truncate operations are unable to execute concurrently. When this occurs, you can set DB2_TRUNCATE_REUSESTORAGE to "IMPORT" or "import", and physical truncation of the object, including data, indexes, long fields, large objects and block maps (for multi-dimensional clustering tables), is skipped and only logical truncation is performed. That is, the IMPORT with REPLACE command empties the table, causing the object's logical size to decrease, but the storage on disk remains allocated.

  This registry variable is dynamic; you can set it or unset it without having to stop and start instance. You can set DB2_TRUNCATE_REUSESTORAGE before an online backup starts and then unset it after online backup completes. For multi-partitioned environments, the registry variable will only be active on the nodes on which the variable is set. DB2_TRUNCATE_REUSESTORAGE is only effective on DMS permanent objects.

  In SAP environments, when DB2_WORKLOAD=SAP is set, the default value of this registry variable is IMPORT.

DB2_USE_DB2JCCT2_JROUTINE

- Operating system: All
- Default=null (not set), Values: OFF/NO/0/FALSE or ON/YES/1/TRUE
- Setting DB2_USE_DB2JCCT2_JROUTINE to any of ON, YES, 1, or TRUE causes the IBM DB2 Driver for JDBC and SQLJ to be used to serve SQL requests for Java routines. Otherwise, the DB2 JDBC Type 2 Driver for Linux, UNIX, and Windows will be used.

DB2_UTIL_MSGPATH

- Operating system: All
- Default=*instanceName*/tmp directory
- The DB2_UTIL_MSGPATH registry variable is used in conjunction with the SYSPROC.ADMIN_CMD procedure, the SYSPROC.ADMIN_REMOVE_MSGS procedure, and the SYSPROC.ADMIN_GET_MSGS UDF. It applies on the instance level. DB2_UTIL_MSGPATH can be set to indicate a directory path on the server where the fenced user ID can read, write and delete files. This directory must be accessible from all coordinator partitions, and must have sufficient space to contain utility message files.

  If this path is not set, the *instanceName*/tmp directory is used as the default (note that *instanceName*/tmp is cleaned up when DB2 is uninstalled).

  If this path is changed, the files that existed in the directory pointed to by the previous setting are not automatically moved or deleted. If you want to retrieve the contents of the messages created under the old path,

you must manually move these messages (which are prefixed with the utility name and postfixed with the user ID) to the new directory to which DB2_UTIL_MSGPATH points. The next utility message file is created, read and cleaned up in the new location.

The files under the DB2_UTIL_MSGPATH directory are utility specific, not transaction dependent. They are not part of the backup image. The files under the DB2_UTIL_MSGPATH directory are user managed; that means a user can delete the message files using the SYSPROC.ADMIN_REMOVE_UTILMSG procedure. These files are not cleaned up by uninstalling DB2.

DB2_VENDOR_INI

- Operating system: AIX, HP-UX, Solaris, and Windows
- Default=null, Values: Any valid path and file.
- Points to a file containing all vendor-specific environment settings. The value is read when the database manager starts.

DB2_XBSA_LIBRARY

- Operating system: AIX, HP-UX, Solaris, and Windows
- Default=null, Values: Any valid path and file.
- Points to the vendor-supplied XBSA library. On AIX, the setting must include the shared object if it is not named shr.o. HP-UX, Solaris, and Windows do not require the shared object name. For example, to use Legato's NetWorker Business Suite Module for DB2, the registry variable must be set as follows:

  ```
  db2set DB2_XSBA_LIBRARY="/usr/lib/libxdb2.a(bsashr10.o)"
  ```

  The XBSA interface can be invoked through the BACKUP DATABASE or the RESTORE DATABASE commands. For example:

  ```
  db2 backup db sample use XBSA
  db2 restore db sample use XBSA
  ```

Related concepts:

- "DB2 registry and environment variables" on page 531
- "Connection concentrator" in *DB2 Connect User's Guide*

Appendix C. Explain tables

Explain tables

The Explain tables capture access plans when the Explain facility is activated. The Explain tables must be created before Explain can be invoked. You can create them using the documented table definitions, or you can create them by invoking the sample command line processor (CLP) script provided in the EXPLAIN.DDL file located in the misc subdirectory of the sqllib directory. To invoke the script, connect to the database where the Explain tables are required, then issue the command:

```
db2 -tf EXPLAIN.DDL
```

The Explain facility uses the following IDs as the schema when qualifying Explain tables that it is populating:

- The session authorization ID for dynamic SQL
- The statement authorization ID for static SQL

The schema can be associated with a set of Explain tables, or aliases that point to a set of Explain tables under a different schema. If no Explain tables are found under the schema, the Explain facility checks for Explain tables under the SYSTOOLS schema and attempts to use those tables.

The population of the Explain tables by the Explain facility will not activate triggers or referential or check constraints. For example, if an insert trigger were defined on the EXPLAIN_INSTANCE table, and an eligible statement were explained, the trigger would not be activated.

To improve the performance of the Explain facility in a partitioned database system, it is recommended that the Explain tables be created in a single partition database partition group, preferably on the same database partition to which you will be connected when compiling the query.

Related reference:

- "ADVISE_INDEX table" on page 608
- "ADVISE_INSTANCE table" on page 611
- "ADVISE_MQT table" on page 612
- "ADVISE_PARTITION table" on page 613
- "ADVISE_TABLE table" on page 614
- "ADVISE_WORKLOAD table" on page 615
- "EXPLAIN_ARGUMENT table" on page 586
- "EXPLAIN_INSTANCE table" on page 593
- "EXPLAIN_OBJECT table" on page 596
- "EXPLAIN_OPERATOR table" on page 599
- "EXPLAIN_PREDICATE table" on page 601
- "EXPLAIN_STATEMENT table" on page 604
- "EXPLAIN_STREAM table" on page 606

EXPLAIN_ARGUMENT table

The EXPLAIN_ARGUMENT table represents the unique characteristics for each individual operator, if there are any.

Table 78. EXPLAIN_ARGUMENT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this Explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this Explain information is related.
OPERATOR_ID	INTEGER	No	No	Unique ID for this operator within this query.
ARGUMENT_TYPE	CHAR(8)	No	No	The type of argument for this operator.
ARGUMENT_VALUE	VARCHAR(1024)	Yes	No	The value of the argument for this operator. NULL if the value is in LONG_ARGUMENT_VALUE.
LONG_ARGUMENT_VALUE	CLOB(2M)	Yes	No	The value of the argument for this operator, when the text will not fit in ARGUMENT_VALUE. NULL if the value is in ARGUMENT_VALUE.

Table 79. ARGUMENT_TYPE and ARGUMENT_VALUE column values

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
AGGMODE	COMPLETE PARTIAL INTERMEDIATE FINAL	Partial aggregation indicators.
BITFLTR	INTEGER FALSE	Size of bit filter used by hash join.
BLD_LEVEL	DB2 Build Identifier	Internal identification string for source code version.
BLKLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT SHARE NONE SHARE UPDATE	Block level lock intent.
CSERQY	TRUE FALSE	Remote query is a common subexpression.
CSETEMP	TRUE FALSE	Temporary Table over Common Subexpression Flag.
DIRECT	TRUE	Direct fetch indicator.

Table 79. ARGUMENT_TYPE and ARGUMENT_VALUE column values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
DPESTFLG	TRUE FALSE	Indicates whether or not the DPNUMPRT value is based on an estimate. Possible values are 'TRUE' (DPNUMPRT represents the estimated number of accessed data partitions) or 'FALSE' (DPNUMPRT represents the actual number of accessed data partitions).
DPLSTPRT	NONE CHARACTER	Represents accessed data partitions. It is a comma-delimited list (for example: 1,3,5) or a hyphenated list (for example: 1-5) of accessed data partitions. A value of 'NONE' means that no data partition remains after specified predicates have been applied.
DPNUMPRT	INTEGER	Represents the actual or estimated number of data partitions accessed.
DSTSEVER	Server name	Destination (ship from) server.
DUPLWARN	TRUE FALSE	Duplicates Warning flag.
EARLYOUT	LEFT RIGHT NONE	Early out indicator.
ENVVAR	Each row of this type will contain: • Environment variable name • Environment variable value	Environment variable affecting the optimizer
ERRTOL	Each row of this type will contain an SQLSTATE and SQLCODE pair.	A list of errors to be tolerated.
FETCHMAX	IGNORE INTEGER	Override value for MAXPAGES argument on FETCH operator.
GREEDY	TRUE	Indicates optimizer used greedy algorithm to plan access.
GLOBLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT NONE INTENT SHARE NO LOCK OBTAINED SHARE SHARE INTENT EXCLUSIVE SUPER EXCLUSIVE UPDATE	Represents global lock intent information for a partitioned table object.
GROUPBYC	TRUE FALSE	Whether Group By columns were provided.
GROUPBYN	Integer	Number of comparison columns.
GROUPBYR	Each row of this type will contain: • Ordinal value of column in group by clause (followed by a colon and a space) • Name of column	Group By requirement.
HASHCODE	24 32	Size (in bits) of hash code used for hash join.

Table 79. ARGUMENT_TYPE and ARGUMENT_VALUE column values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
INNERCOL	Each row of this type will contain: • Ordinal value of column in order (followed by a colon and a space) • Name of column • Order value **(A)** Ascending **(D)** Descending	Inner order columns.
INPUTXID	A context node identifier	INPUTXID identifies the input context node used by the XSCAN operator.
ISCANMAX	IGNORE INTEGER	Override value for MAXPAGES argument on ISCAN operator.
JN_INPUT	INNER OUTER	Indicates if operator is the operator feeding the inner or outer of a join.
LISTENER	TRUE FALSE	Listener Table Queue indicator.
MAXPAGES	ALL NONE INTEGER	Maximum pages expected for Prefetch.
MAXRIDS	NONE INTEGER	Maximum Row Identifiers to be included in each list prefetch request.
NUMROWS	INTEGER	Number of rows expected to be sorted.
ONEFETCH	TRUE FALSE	One Fetch indicator.
OUTERCOL	Each row of this type will contain: • Ordinal value of column in order (followed by a colon and a space) • Name of column • Order value **(A)** Ascending **(D)** Descending	Outer order columns.
OUTERJN	LEFT RIGHT FULL LEFT (ANTI) RIGHT (ANTI)	Outer join indicator.
PARTCOLS	Name of Column	Partitioning columns for operator.
PREFETCH	LIST NONE SEQUENTIAL	Type of Prefetch Eligible.
REOPT	ALWAYS ONCE	The statement is optimized using bind-in values for parameter markers, host variables, and special registers.
RMTQTEXT	Query text	Remote Query Text
RNG_PROD	Function name	Range producing function for extended index access.

Table 79. ARGUMENT_TYPE and ARGUMENT_VALUE column values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
ROWLOCK	EXCLUSIVE NONE REUSE SHARE SHORT (INSTANT) SHARE UPDATE	Row Lock Intent.
ROWWIDTH	INTEGER	Width of row to be sorted.
RSUFFIX	Query text	Remote SQL suffix.
SCANDIR	FORWARD REVERSE	Scan Direction.
SCANGRAN	INTEGER	Intra-partition parallelism, granularity of the intra-partition parallel scan, expressed in SCANUNITs.
SCANTYPE	LOCAL PARALLEL	intra-partition parallelism, Index or Table scan.
SCANUNIT	ROW PAGE	Intra-partition parallelism, scan granularity unit.
SHARED	TRUE	Intra-partition parallelism, shared TEMP indicator.
SLOWMAT	TRUE FALSE	Slow Materialization flag.
SNGLPROD	TRUE FALSE	Intra-partition parallelism sort or temp produced by a single agent.
SORTKEY	Each row of this type will contain: • Ordinal value of column in key (followed by a colon and a space) • Name of column • Order value **(A)** Ascending **(D)** Descending	Sort key columns.
SORTTYPE	PARTITIONED SHARED ROUND ROBIN REPLICATED	Intra-partition parallelism, sort type.
SRCSEVER	Server name	Source (ship to) server.
SPILLED	INTEGER	Estimated number of pages in SORT spill.
SQLCA	Warning information	Warnings and reason codes issued during Explain operation.
STMTHEAP	INTEGER	Size of statement heap at start of statement compile.
STREAM	TRUE FALSE	Remote source is streaming.
TABLOCK	EXCLUSIVE INTENT EXCLUSIVE INTENT NONE INTENT SHARE REUSE SHARE SHARE INTENT EXCLUSIVE SUPER EXCLUSIVE UPDATE	Table Lock Intent.
TEMPSIZE	INTEGER	Temporary table page size.
TQDEGREE	INTEGER	Intra-partition parallelism, number of subagents accessing Table Queue.

Table 79. ARGUMENT_TYPE and ARGUMENT_VALUE column values (continued)

ARGUMENT_TYPE Value	Possible ARGUMENT_VALUE Values	Description
TQMERGE	TRUE FALSE	Merging (sorted) Table Queue indicator.
TQREAD	READ AHEAD STEPPING SUBQUERY STEPPING	Table Queue reading property.
TQSEND	BROADCAST DIRECTED SCATTER SUBQUERY DIRECTED	Table Queue send property.
TQTYPE	LOCAL	Intra-partition parallelism, Table Queue.
TQ_ORIGIN	ASYNCHRONY	The reason that Table Queue was introduced into the access plan.
TRUNCSRT	TRUE	Truncated sort (limits number of rows produced).
UNIQUE	TRUE FALSE	Uniqueness indicator.
UNIQKEY	Each row of this type will contain: • Ordinal value of column in key (followed by a colon and a space) • Name of Column	Unique key columns.
VOLATILE	TRUE	Volatile table
XDFOUT	DECIMAL	XDFOUT indicates the expected number of documents to be returned by the XISCAN operator for each context node.
XLOGID	An identifier consisting of an SQL schema name and the name of an index over XML data	XLOGID identifies the index over XML data chosen by the optimizer for the XISCAN operator.
XPATH	An XPATH expression and result set in an internal format	This argument indicates the evaluation of an XPATH expression by the XSCAN operator.
XPHYID	An identifier consisting of an SQL schema name and the name of a physical index over XML data	XPHYID identifies the physical index that is associated with an index over XML data used by the XISCAN operator.

EXPLAIN_DIAGNOSTIC table

The EXPLAIN_DIAGNOSTIC table contains an entry for each diagnostic message produced for a particular instance of an explained statement in the EXPLAIN_STATEMENT table.

The EXPLAIN_GET_MSGS table function queries the EXPLAIN_DIAGNOSTIC and EXPLAIN_DIAGNOSTIC_DATA Explain tables and returns formatted messages.

Table 80. EXPLAIN_DIAGNOSTIC Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK, FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK, FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	PK, FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	PK, FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	PK, FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	PK, FK	Level of Explain information for which this row is relevant. Valid values are: **O** Original text (as entered by user) **P** PLAN SELECTION
STMTNO	INTEGER	No	PK, FK	Statement number within package to which this Explain information is related. Set to 1 for dynamic Explain SQL statements. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS system catalog view.
SECTNO	INTEGER	No	PK, FK	Section number within package that contains this SQL statement. For dynamic Explain SQL statements, this is the section number used to hold the section for this statement at run time. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS system catalog view.
DIAGNOSTIC_ID	INTEGER	No	PK	ID of the diagnostic for a particular instance of a statement in the EXPLAIN_STATEMENT table.
CODE	INTEGER	No	No	A unique number assigned to each diagnostic message. The number can be used by a message API to retrieve the full text of the diagnostic message.

Related reference:

- "EXPLAIN_DIAGNOSTIC_DATA table" on page 592
- "EXPLAIN_GET_MSGS table function" in *Administrative SQL Routines and Views*
- "EXPLAIN_STATEMENT table" on page 604
- "SYSCAT.STATEMENTS catalog view" in *SQL Reference, Volume 1*

EXPLAIN_DIAGNOSTIC_DATA table

The EXPLAIN_DIAGNOSTIC_DATA table contains message tokens for specific diagnostic messages that are recorded in the EXPLAIN_DIAGNOSTIC table. The message tokens provide additional information that is specific to the execution of the SQL statement that generated the message.

The EXPLAIN_GET_MSGS table function queries the EXPLAIN_DIAGNOSTIC and EXPLAIN_DIAGNOSTIC_DATA Explain tables, and returns formatted messages.

Table 81. EXPLAIN_DIAGNOSTIC_DATA Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant. Valid values are: **O** Original text (as entered by user) **P** PLAN SELECTION
STMTNO	INTEGER	No	FK	Statement number within package to which this Explain information is related. Set to 1 for dynamic Explain SQL statements. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS system catalog view.
SECTNO	INTEGER	No	FK	Section number within package that contains this SQL statement. For dynamic Explain SQL statements, this is the section number used to hold the section for this statement at run time. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS system catalog view.
DIAGNOSTIC_ID	INTEGER	No	PK	ID of the diagnostic for a particular instance of a statement in the EXPLAIN_STATEMENT table.
ORDINAL	INTEGER	No	No	Position of token in the full message text.
TOKEN	VARCHAR(1000)	Yes	No	Message token to be inserted into the full message text; might be truncated.
TOKEN_LONG	BLOB(3M)	Yes	No	More detailed information, if available.

Related reference:

- "EXPLAIN_DIAGNOSTIC table" on page 591
- "EXPLAIN_GET_MSGS table function" in *Administrative SQL Routines and Views*
- "EXPLAIN_STATEMENT table" on page 604
- "SYSCAT.STATEMENTS catalog view" in *SQL Reference, Volume 1*

EXPLAIN_INSTANCE table

The EXPLAIN_INSTANCE table is the main control table for all Explain information. Each row of data in the Explain tables is explicitly linked to one unique row in this table. The EXPLAIN_INSTANCE table gives basic information about the source of the SQL statements being explained as well as information about the environment in which the explanation took place.

Table 82. EXPLAIN_INSTANCE Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	PK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	PK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	PK	Version of the source of the Explain request.
EXPLAIN_OPTION	CHAR(1)	No	No	Indicates what Explain Information was requested for this request. Possible values are: **P** PLAN SELECTION
SNAPSHOT_TAKEN	CHAR(1)	No	No	Indicates whether an Explain Snapshot was taken for this request. Possible values are: **Y** Yes, an Explain Snapshot(s) was taken and stored in the EXPLAIN_STATEMENT table. Regular Explain information was also captured. **N** No Explain Snapshot was taken. Regular Explain information was captured. **O** Only an Explain Snapshot was taken. Regular Explain information was not captured.
DB2_VERSION	CHAR(7)	No	No	Release number for the DB2 product that processed this explain request. Format is *vv.rr.m*, where: **vv** Version number **rr** Release number **m** Maintenance release number
SQL_TYPE	CHAR(1)	No	No	Indicates whether the Explain Instance was for static or dynamic SQL. Possible values are: **S** Static SQL **D** Dynamic SQL
QUERYOPT	INTEGER	No	No	Indicates the query optimization class used by the SQL Compiler at the time of the Explain invocation. The value indicates what level of query optimization was performed by the SQL Compiler for the SQL statements being explained.

Table 82. EXPLAIN_INSTANCE Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
BLOCK	CHAR(1)	No	No	Indicates what type of cursor blocking was used when compiling the SQL statements. For more information, see the BLOCK column in SYSCAT.PACKAGES. Possible values are: **N** No Blocking **U** Block Unambiguous Cursors **B** Block All Cursors
ISOLATION	CHAR(2)	No	No	Indicates what type of isolation was used when compiling the SQL statements. For more information, see the ISOLATION column in SYSCAT.PACKAGES. Possible values are: **RR** Repeatable Read **RS** Read Stability **CS** Cursor Stability **UR** Uncommitted Read
BUFFPAGE	INTEGER	No	No	Contains the value of the BUFFPAGE database configuration setting at the time of the Explain invocation.
AVG_APPLS	INTEGER	No	No	Contains the value of the AVG_APPLS configuration parameter at the time of the Explain invocation.
SORTHEAP	INTEGER	No	No	Contains the value of the SORTHEAP database configuration setting at the time of the Explain invocation.
LOCKLIST	INTEGER	No	No	Contains the value of the LOCKLIST database configuration setting at the time of the Explain invocation.
MAXLOCKS	SMALLINT	No	No	Contains the value of the MAXLOCKS database configuration setting at the time of the Explain invocation.
LOCKS_AVAIL	INTEGER	No	No	Contains the number of locks assumed to be available by the optimizer for each user. (Derived from LOCKLIST and MAXLOCKS.)
CPU_SPEED	DOUBLE	No	No	Contains the value of the CPUSPEED database manager configuration setting at the time of the Explain invocation.
REMARKS	VARCHAR(254)	Yes	No	User-provided comment.
DBHEAP	INTEGER	No	No	Contains the value of the DBHEAP database configuration setting at the time of Explain invocation.
COMM_SPEED	DOUBLE	No	No	Contains the value of the COMM_BANDWIDTH database configuration setting at the time of Explain invocation.
PARALLELISM	CHAR(2)	No	No	Possible values are: • N = No parallelism • P = Intra-partition parallelism • IP = Inter-partition parallelism • BP = Intra-partition parallelism and inter-partition parallelism
DATAJOINER	CHAR(1)	No	No	Possible values are: • N = Non-federated systems plan • Y = Federated systems plan

EXPLAIN_OBJECT table

The EXPLAIN_OBJECT table identifies those data objects required by the access plan generated to satisfy the SQL statement.

Table 83. EXPLAIN_OBJECT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OBJECT_SCHEMA	VARCHAR(128)	No	No	Schema to which this object belongs.
OBJECT_NAME	VARCHAR(128)	No	No	Name of the object.
OBJECT_TYPE	CHAR(2)	No	No	Descriptive label for the type of object.
CREATE_TIME	TIMESTAMP	Yes	No	Time of Object's creation; null if a table function.
STATISTICS_TIME	TIMESTAMP	Yes	No	Last time of update to statistics for this object; null if statistics do not exist for this object.
COLUMN_COUNT	SMALLINT	No	No	Number of columns in this object.
ROW_COUNT	INTEGER	No	No	Estimated number of rows in this object.
WIDTH	INTEGER	No	No	The average width of the object in bytes. Set to -1 for an index.
PAGES	BIGINT	No	No	Estimated number of pages that the object occupies in the buffer pool. Set to -1 for a table function.
DISTINCT	CHAR(1)	No	No	Indicates whether the rows in the object are distinct (that is, whether there are duplicates). Possible values are: **Y** Yes **N** No
TABLESPACE_NAME	VARCHAR(128)	Yes	No	Name of the table space in which this object is stored; set to null if no table space is involved.
OVERHEAD	DOUBLE	No	No	Total estimated overhead, in milliseconds, for a single random I/O to the specified table space. Includes controller overhead, disk seek, and latency times. Set to -1 if no table space is involved.
TRANSFER_RATE	DOUBLE	No	No	Estimated time to read a data page, in milliseconds, from the specified table space. Set to -1 if no table space is involved.
PREFETCHSIZE	INTEGER	No	No	Number of data pages to be read when prefetch is performed. Set to -1 for a table function.
EXTENTSIZE	INTEGER	No	No	Size of extent, in data pages. This many pages are written to one container in the table space before switching to the next container. Set to -1 for a table function.

Table 83. EXPLAIN_OBJECT Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
CLUSTER	DOUBLE	No	No	Degree of data clustering with the index. If >= 1, this is the CLUSTERRATIO. If >= 0 and < 1, this is the CLUSTERFACTOR. Set to -1 for a table, table function, or if this statistic is not available.
NLEAF	BIGINT	No	No	Number of leaf pages this index object's values occupy. Set to -1 for a table, table function, or if this statistic is not available.
NLEVELS	INTEGER	No	No	Number of index levels in this index object's tree. Set to -1 for a table, table function, or if this statistic is not available.
FULLKEYCARD	BIGINT	No	No	Number of distinct full key values contained in this index object. Set to -1 for a table, table function, or if this statistic is not available.
OVERFLOW	BIGINT	No	No	Total number of overflow records in the table. Set to -1 for an index, table function, or if this statistic is not available.
FIRSTKEYCARD	BIGINT	No	No	Number of distinct first key values. Set to –1 for a table, table function, or if this statistic is not available.
FIRST2KEYCARD	BIGINT	No	No	Number of distinct first key values using the first {2,3,4} columns of the index. Set to –1 for a table, table function, or if this statistic is not available.
FIRST3KEYCARD	BIGINT	No	No	
FIRST4KEYCARD	BIGINT	No	No	
SEQUENTIAL_PAGES	BIGINT	No	No	Number of leaf pages located on disk in index key order with few or no large gaps between them. Set to –1 for a table, table function, or if this statistic is not available.
DENSITY	INTEGER	No	No	Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percentage (integer between 0 and 100). Set to –1 for a table, table function, or if this statistic is not available.
STATS_SRC	CHAR(1)	No	No	Indicates the source for the statistics. Set to 1 if from single node.
AVERAGE_SEQUENCE_GAP	DOUBLE	No	No	Gap between sequences.
AVERAGE_SEQUENCE_FETCH_GAP	DOUBLE	No	No	Gap between sequences when fetching using the index.
AVERAGE_SEQUENCE_PAGES	DOUBLE	No	No	Average number of index pages accessible in sequence.
AVERAGE_SEQUENCE_FETCH_PAGES	DOUBLE	No	No	Average number of table pages accessible in sequence when fetching using the index.
AVERAGE_RANDOM_PAGES	DOUBLE	No	No	Average number of random index pages between sequential page accesses.
AVERAGE_RANDOM_FETCH_PAGES	DOUBLE	No	No	Average number of random table pages between sequential page accesses when fetching using the index.
NUMRIDS	BIGINT	No	No	Total number of row identifiers in the index.
NUMRIDS_DELETED	BIGINT	No	No	Total number of psuedo-deleted row identifiers in the index.
NUM_EMPTY_LEAFS	BIGINT	No	No	Total number of empty leaf pages in the index.
ACTIVE_BLOCKS	BIGINT	No	No	Total number of active multidimensional clustering (MDC) blocks in the table.
NUM_DATA_PART	INTEGER	No	No	Number of data partitions for a partitioned table. Set to 1 if the table is not partitioned.

Table 84. Possible OBJECT_TYPE Values

Value	Description
IX	Index
NK	Nickname
RX	RCT Index
DP_TABLE	Data partitioned table
TA	Table
TF	Table Function
+A	Compiler-referenced Alias
+C	Compiler-referenced Constraint
+F	Compiler-referenced Function
+G	Compiler-referenced Trigger
+N	Compiler-referenced Nickname
+T	Compiler-referenced Table
+V	Compiler-referenced View

EXPLAIN_OPERATOR table

The EXPLAIN_OPERATOR table contains all the operators needed to satisfy the query statement by the query compiler.

Table 85. EXPLAIN_OPERATOR Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	INTEGER	No	No	Unique ID for this operator within this query.
OPERATOR_TYPE	CHAR(6)	No	No	Descriptive label for the type of operator.
TOTAL_COST	DOUBLE	No	No	Estimated cumulative total cost (in timerons) of executing the chosen access plan up to and including this operator.
IO_COST	DOUBLE	No	No	Estimated cumulative I/O cost (in data page I/Os) of executing the chosen access plan up to and including this operator.
CPU_COST	DOUBLE	No	No	Estimated cumulative CPU cost (in instructions) of executing the chosen access plan up to and including this operator.
FIRST_ROW_COST	DOUBLE	No	No	Estimated cumulative cost (in timerons) of fetching the first row for the access plan up to and including this operator. This value includes any initial overhead required.
RE_TOTAL_COST	DOUBLE	No	No	Estimated cumulative cost (in timerons) of fetching the next row for the chosen access plan up to and including this operator.
RE_IO_COST	DOUBLE	No	No	Estimated cumulative I/O cost (in data page I/Os) of fetching the next row for the chosen access plan up to and including this operator.
RE_CPU_COST	DOUBLE	No	No	Estimated cumulative CPU cost (in instructions) of fetching the next row for the chosen access plan up to and including this operator.
COMM_COST	DOUBLE	No	No	Estimated cumulative communication cost (in TCP/IP frames) of executing the chosen access plan up to and including this operator.
FIRST_COMM_COST	DOUBLE	No	No	Estimated cumulative communications cost (in TCP/IP frames) of fetching the first row for the chosen access plan up to and including this operator. This value includes any initial overhead required.
BUFFERS	DOUBLE	No	No	Estimated buffer requirements for this operator and its inputs.
REMOTE_TOTAL_COST	DOUBLE	No	No	Estimated cumulative total cost (in timerons) of performing operation(s) on remote database(s).

Table 85. EXPLAIN_OPERATOR Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
REMOTE_COMM_COST	DOUBLE	No	No	Estimated cumulative communication cost of executing the chosen remote access plan up to and including this operator.

Table 86. OPERATOR_TYPE values

Value	Description
DELETE	Delete
EISCAN	Extended Index Scan
FETCH	Fetch
FILTER	Filter rows
GENROW	Generate Row
GRPBY	Group By
HSJOIN	Hash Join
INSERT	Insert
IXAND	Dynamic Bitmap Index ANDing
IXSCAN	Relational index scan
MSJOIN	Merge Scan Join
NLJOIN	Nested loop Join
RETURN	Result
RIDSCN	Row Identifier (RID) Scan
SHIP	Ship query to remote system
SORT	Sort
TBSCAN	Table Scan
TEMP	Temporary Table Construction
TQ	Table Queue
UNION	Union
UNIQUE	Duplicate Elimination
UPDATE	Update
XISCAN	Index scan over XML data
XSCAN	XML document navigation scan
XANDOR	Index ANDing and ORing over XML data
XITERATE	XML sequence iterator
XUNNEST	XML sequence unnest

EXPLAIN_PREDICATE table

The EXPLAIN_PREDICATE table identifies which predicates are applied by a specific operator.

Table 87. EXPLAIN_PREDICATE Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
OPERATOR_ID	INTEGER	No	No	Unique ID for this operator within this query.
PREDICATE_ID	INTEGER	No	No	Unique ID for this predicate for the specified operator. A value of ″-1″ is shown for operator predicates constructed by the Explain tool which are not optimizer objects and do not exist in the optimizer plan.
HOW_APPLIED	CHAR(10)	No	No	How predicate is being used by the specified operator.
WHEN_EVALUATED	CHAR(3)	No	No	Indicates when the subquery used in this predicate is evaluated. Possible values are: **blank** This predicate does not contain a subquery. **EAA** The subquery used in this predicate is evaluated at application (EAA). That is, it is re-evaluated for every row processed by the specified operator, as the predicate is being applied. **EAO** The subquery used in this predicate is evaluated at open (EAO). That is, it is re-evaluated only once for the specified operator, and its results are re-used in the application of the predicate for each row. **MUL** There is more than one type of subquery in this predicate.
RELOP_TYPE	CHAR(2)	No	No	The type of relational operator used in this predicate.
SUBQUERY	CHAR(1)	No	No	Whether or not a data stream from a subquery is required for this predicate. There may be multiple subquery streams required. Possible values are: **N** No subquery stream is required **Y** One or more subquery streams is required

Table 87. EXPLAIN_PREDICATE Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
FILTER_FACTOR	DOUBLE	No	No	The estimated fraction of rows that will be qualified by this predicate. A value of ″-1″ is shown when FILTER_FACTOR is not applicable. FILTER_FACTOR is not applicable for operator predicates constructed by the Explain tool which are not optimizer objects and do not exist in the optimizer plan.
PREDICATE_TEXT	CLOB(2M)	Yes	No	The text of the predicate as recreated from the internal representation of the SQL or XQuery statement. If the value of a host variable, special register, or parameter marker is used during compilation of the statement, this value will appear at the end of the predicate text enclosed in brackets. The value will be stored in the EXPLAIN_PREDICATE table only if the statement is executed by a user who has DBADM authority, or if the DB2 registry variable DB2_VIEW_REOPT_VALUES is set to YES; otherwise, empty brackets will appear at the end of the predicate text. Null if not available.
RANGE_NUM	INTEGER	Yes	No	Range of data partition elimination predicates, which enables the grouping of predicates that are used for data partition elimination by range. Null value for all other predicate types.

Table 88. Possible HOW_APPLIED Values

Value	Description
BSARG	Evaluated as a sargable predicate once for every block
DPSTART	Start key predicate used in data partition elimination
DPSTOP	Stop key predicate used in data partition elimination
JOIN	Used to join tables
RESID	Evaluated as a residual predicate
SARG	Evaluated as a sargable predicate for index or data page
START	Used as a start condition
STOP	Used as a stop condition

Table 89. Possible RELOP_TYPE Values

Value	Description
blanks	Not Applicable
EQ	Equals
GE	Greater Than or Equal
GT	Greater Than
IN	In list
LE	Less Than or Equal
LK	Like
LT	Less Than
NE	Not Equal
NL	Is Null
NN	Is Not Null

EXPLAIN_STATEMENT table

The EXPLAIN_STATEMENT table contains the text of the SQL statement as it exists for the different levels of Explain information. The original SQL statement as entered by the user is stored in this table along with the version used (by the optimizer) to choose an access plan to satisfy the SQL statement. The latter version may bear little resemblance to the original as it may have been rewritten and/or enhanced with additional predicates as determined by the SQL Compiler.

Table 90. EXPLAIN_STATEMENT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	PK, FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	PK, FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	PK, FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	PK, FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	PK	Level of Explain information for which this row is relevant. Valid values are: **O** Original Text (as entered by user) **P** PLAN SELECTION
STMTNO	INTEGER	No	PK	Statement number within package to which this explain information is related. Set to 1 for dynamic Explain SQL statements. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS catalog view.
SECTNO	INTEGER	No	PK	Section number within package that contains this SQL statement. For dynamic Explain SQL statements, this is the section number used to hold the section for this statement at runtime. For static SQL statements, this value is the same as the value used for the SYSCAT.STATEMENTS catalog view.
QUERYNO	INTEGER	No	No	Numeric identifier for explained SQL statement. For dynamic SQL statements (excluding the EXPLAIN SQL statement) issued through CLP or CLI, the default value is a sequentially incremented value. Otherwise, the default value is the value of STMTNO for static SQL statements and 1 for dynamic SQL statements.
QUERYTAG	CHAR(20)	No	No	Identifier tag for each explained SQL statement. For dynamic SQL statements issued through CLP (excluding the EXPLAIN SQL statement), the default value is 'CLP'. For dynamic SQL statements issued through CLI (excluding the EXPLAIN SQL statement), the default value is 'CLI'. Otherwise, the default value used is blanks.
STATEMENT_TYPE	CHAR(2)	No	No	Descriptive label for type of query being explained. Possible values are: **S** Select **D** Delete **DC** Delete where current of cursor **I** Insert **U** Update **UC** Update where current of cursor

Table 90. EXPLAIN_STATEMENT Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
UPDATABLE	CHAR(1)	No	No	Indicates if this statement is considered updatable. This is particularly relevant to SELECT statements which may be determined to be potentially updatable. Possible values are: ' ' Not applicable (blank) **N** No **Y** Yes
DELETABLE	CHAR(1)	No	No	Indicates if this statement is considered deletable. This is particularly relevant to SELECT statements which may be determined to be potentially deletable. Possible values are: ' ' Not applicable (blank) **N** No **Y** Yes
TOTAL_COST	DOUBLE	No	No	Estimated total cost (in timerons) of executing the chosen access plan for this statement; set to 0 (zero) if EXPLAIN_LEVEL is *O* (original text) since no access plan has been chosen at this time.
STATEMENT_TEXT	CLOB(2M)	No	No	Text or portion of the text of the SQL statement being explained. The text shown for the Plan Selection level of Explain has been reconstructed from the internal representation and is SQL-like in nature; that is, the reconstructed statement is not guaranteed to follow correct SQL syntax.
SNAPSHOT	BLOB(10M)	Yes	No	Snapshot of internal representation for this SQL statement at the Explain_Level shown. This column is intended for use with DB2 Visual Explain. Column is set to null if EXPLAIN_LEVEL is *0* (original statement) since no access plan has been chosen at the time that this specific version of the statement is captured.
QUERY_DEGREE	INTEGER	No	No	Indicates the degree of intra-partition parallelism at the time of Explain invocation. For the original statement, this contains the directed degree of intra-partition parallelism. For the PLAN SELECTION, this contains the degree of intra-partition parallelism generated for the plan to use.

EXPLAIN_STREAM table

The EXPLAIN_STREAM table represents the input and output data streams between individual operators and data objects. The data objects themselves are represented in the EXPLAIN_OBJECT table. The operators involved in a data stream are to be found in the EXPLAIN_OPERATOR table.

Table 91. EXPLAIN_STREAM Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	FK	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	FK	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	FK	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	FK	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	FK	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	FK	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	FK	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	FK	Section number within package to which this explain information is related.
STREAM_ID	INTEGER	No	No	Unique ID for this data stream within the specified operator.
SOURCE_TYPE	CHAR(1)	No	No	Indicates the source of this data stream: **O** Operator **D** Data Object
SOURCE_ID	SMALLINT	No	No	Unique ID for the operator within this query that is the source of this data stream. Set to -1 if SOURCE_TYPE is 'D'.
TARGET_TYPE	CHAR(1)	No	No	Indicates the target of this data stream: **O** Operator **D** Data Object
TARGET_ID	SMALLINT	No	No	Unique ID for the operator within this query that is the target of this data stream. Set to -1 if TARGET_TYPE is 'D'.
OBJECT_SCHEMA	VARCHAR(128)	Yes	No	Schema to which the affected data object belongs. Set to null if both SOURCE_TYPE and TARGET_TYPE are 'O'.
OBJECT_NAME	VARCHAR(128)	Yes	No	Name of the object that is the subject of data stream. Set to null if both SOURCE_TYPE and TARGET_TYPE are 'O'.
STREAM_COUNT	DOUBLE	No	No	Estimated cardinality of data stream.
COLUMN_COUNT	SMALLINT	No	No	Number of columns in data stream.
PREDICATE_ID	INTEGER	No	No	If this stream is part of a subquery for a predicate, the predicate ID will be reflected here, otherwise the column is set to -1.

Table 91. EXPLAIN_STREAM Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
COLUMN_NAMES	CLOB(2M)	Yes	No	This column contains the names and ordering information of the columns involved in this stream. These names will be in the format of: `NAME1(A)+NAME2(D)+NAME3+NAME4` Where *(A)* indicates a column in ascending order, *(D)* indicates a column in descending order, and no ordering information indicates that either the column is not ordered or ordering is not relevant.
PMID	SMALLINT	No	No	Distribution map ID.
SINGLE_NODE	CHAR(5)	Yes	No	Indicates whether this data stream is on a single or on multiple database partitions: **MULT** On multiple database partitions **COOR** On coordinator node **HASH** Directed using hashing **RID** Directed using the row ID **FUNC** Directed using a function (HASHEDVALUE() or DBPARTITIONNUM()) **CORR** Directed using a correlation value **Numberic** Directed to predetermined single node
PARTITION_COLUMNS	CLOB(2M)	Yes	No	List of the columns on which this data stream is distributed.
SEQUENCE_SIZES	CLOB(2M)	Yes	No	Lists the expected sequence size for XML columns, or shows ″NA″ (not applicable) for any non-XML columns in the data stream. Set to null if there is not at least one XML column in the data stream.

ADVISE_INDEX table

The ADVISE_INDEX table represents the recommended indexes.

Table 92. ADVISE_INDEX Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	No	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	No	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	No	Name of the package running when the dynamic statement was explained or name of the source file when static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	No	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	No	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	No	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	No	Statement number within package to which this explain information is related.
SECTNO	INTEGER	No	No	Section number within package to which this explain information is related.
QUERYNO	INTEGER	No	No	Numeric identifier for explained SQL statement. For dynamic SQL statements (excluding the EXPLAIN SQL statement) issued through CLP or CLI, the default value is a sequentially incremented value. Otherwise, the default value is the value of STMTNO for static SQL statements and 1 for dynamic SQL statements.
QUERYTAG	CHAR(20)	No	No	Identifier tag for each explained SQL statement. For dynamic SQL statements issued through CLP (excluding the EXPLAIN SQL statement), the default value is 'CLP'. For dynamic SQL statements issued through CLI (excluding the EXPLAIN SQL statement), the default value is 'CLI'. Otherwise, the default value used is blanks.
NAME	VARCHAR(128)	No	No	Name of the index.
CREATOR	VARCHAR(128)	No	No	Qualifier of the index name.
TBNAME	VARCHAR(128)	No	No	Name of the table or nickname on which the index is defined.
TBCREATOR	VARCHAR(128)	No	No	Qualifier of the table name.
COLNAMES	CLOB(2M)	No	No	List of column names.
UNIQUERULE	CHAR(1)	No	No	Unique rule: D = Duplicates allowed P = Primary index U = Unique entries only allowed
COLCOUNT	SMALLINT	No	No	Number of columns in the key plus the number of include columns if any.
IID	SMALLINT	No	No	Internal index ID.
NLEAF	BIGINT	No	No	Number of leaf pages; –1 if statistics are not gathered.
NLEVELS	SMALLINT	No	No	Number of index levels; –1 if statistics are not gathered.
FIRSTKEYCARD	BIGINT	No	No	Number of distinct first key values; –1 if statistics are not gathered.
FULLKEYCARD	BIGINT	No	No	Number of distinct full key values; –1 if statistics are not gathered.

Table 92. ADVISE_INDEX Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
CLUSTERRATIO	SMALLINT	No	No	Degree of data clustering with the index; -1 if statistics are not gathered or if detailed index statistics are gathered (in which case, CLUSTERFACTOR will be used instead).
AVGPARTITION_ CLUSTERRATIO	SMALLINT	No	No	Degree of data clustering within a single data partition. -1 if the table is not table partitioned, if statistics are not gathered, or if detailed statistics are gathered (in which case AVGPARTITION_CLUSTERFACTOR will be used instead).
AVGPARTITION_ CLUSTERFACTOR	DOUBLE	No	No	Finer measurement of the degree of clustering within a single data partition. -1 if the table is not table partitioned, if statistics are not gathered, or if the index is defined on a nickname.
AVGPARTITION_PAGE_ FETCH_PAIRS	VARCHAR(520)	No	No	A list of paired integers in character form. Each pair represents a potential buffer pool size and the corresponding page fetches required to access a single data partition from the table. Zero-length string if no data is available, or if the table is not table partitioned.
DATAPARTITION_ CLUSTERFACTOR	DOUBLE	No	No	A statistic measuring the "clustering" of the index keys with regard to data partitions. This field holds a number between zero and one, with one representing perfect clustering and zero representing no clustering.
CLUSTERFACTOR	DOUBLE	No	No	Finer measurement of degree of clustering, or –1 if detailed index statistics have not been gathered or if the index is defined on a nickname.
USERDEFINED	SMALLINT	No	No	Defined by the user.
SYSTEM_REQUIRED	SMALLINT	No	No	1 if one or the other of the following conditions is met: – This index is required for a primary or unique key constraint, or this index is a dimension block index or composite block index for a multi-dimensional clustering (MDC) table. – This is an index on the (OID) column of a typed table. 2 if both of the following conditions are met: – This index is required for a primary or unique key constraint, or this index is a dimension block index or composite block index for an MDC table. – This is an index on the (OID) column of a typed table. 0 otherwise.
CREATE_TIME	TIMESTAMP	No	No	Time when the index was created.
STATS_TIME	TIMESTAMP	Yes	No	Last time when any change was made to recorded statistics for this index. Null if no statistics available.
PAGE_FETCH_PAIRS	VARCHAR(520)	No	No	A list of pairs of integers, represented in character form. Each pair represents the number of pages in a hypothetical buffer, and the number of page fetches required to scan the table with this index using that hypothetical buffer. (Zero-length string if no data available.)
REMARKS	VARCHAR(254)	Yes	No	User-supplied comment, or null.
DEFINER	VARCHAR(128)	No	No	User who created the index.
CONVERTED	CHAR(1)	No	No	Reserved for future use.

Table 92. ADVISE_INDEX Table (continued). PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
SEQUENTIAL_PAGES	BIGINT	No	No	Number of leaf pages located on disk in index key order with few or no large gaps between them. (–1 if no statistics are available.)
DENSITY	INTEGER	No	No	Ratio of SEQUENTIAL_PAGES to number of pages in the range of pages occupied by the index, expressed as a percent (integer between 0 and 100, –1 if no statistics are available.)
FIRST2KEYCARD	BIGINT	No	No	Number of distinct keys using the first two columns of the index (–1 if no statistics or inapplicable)
FIRST3KEYCARD	BIGINT	No	No	Number of distinct keys using the first three columns of the index (–1 if no statistics or inapplicable)
FIRST4KEYCARD	BIGINT	No	No	Number of distinct keys using the first four columns of the index (–1 if no statistics or inapplicable)
PCTFREE	SMALLINT	No	No	Percentage of each index leaf page to be reserved during initial building of the index. This space is available for future inserts after the index is built.
UNIQUE_COLCOUNT	SMALLINT	No	No	The number of columns required for a unique key. Always <=COLCOUNT. < COLCOUNT only if there a include columns. –1 if index has no unique key (permits duplicates)
MINPCTUSED	SMALLINT	No	No	If not zero, then online index defragmentation is enabled, and the value is the threshold of minimum used space before merging pages.
REVERSE_SCANS	CHAR(1)	No	No	Y = Index supports reverse scans N = Index does not support reverse scans
USE_INDEX	CHAR(1)	Yes	No	Y = index recommended or evaluated N = index not to be recommended R = an existing clustering RID index was recommended (by the Design Advisor) to be unclustered; this is the case when a new clustering RID index is recommended for the table
CREATION_TEXT	CLOB(2M)	No	No	The SQL statement used to create the index.
PACKED_DESC	BLOB(1M)	Yes	No	Internal description of the table.
RUN_ID	TIMESTAMP	Yes	FK	A value corresponding to the START_TIME of a row in the ADVISE_INSTANCE table, linking it to the same Design Advisor run.
INDEXTYPE	VARCHAR(4)	No	No	Type of index. CLUS = Clustering REG = Regular DIM = Dimension block index BLOK = Block index
EXISTS	CHAR(1)	No	No	Set to 'Y' if the index exists in the database catalog.
RIDTOBLOCK	CHAR(1)	No	No	Set to 'Y' if the RID index was used to make a block index in the Design Advisor.

ADVISE_INSTANCE table

The ADVISE_INSTANCE table contains information about db2advis execution, including start time. Contains one row for each execution of db2advis. Other ADVISE tables have a foreign key (RUN_ID) that links to the START_TIME column of the ADVISE_INSTANCE table for rows created during the same Design Advisor run.

Table 93. ADVISE_INSTANCE Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
START_TIME	TIMESTAMP	No	PK	Time at which db2advis execution begins.
END_TIME	TIMESTAMP	No	No	Time at which db2advis execution ends.
MODE	VARCHAR(4)	No	No	The value that was specified with the -m option on the Design Advisor; for example, 'MC' to specify MQT and MDC.
WKLD_COMPRESSION	CHAR(4)	No	No	The workload compression under which the Design Advisor was run.
STATUS	CHAR(9)	No	No	The status of a Design Advisor run. Status can be 'STARTED', 'COMPLETED' (if successful), or an error number that is prefixed by 'EI' for internal errors or 'EX' for external errors, in which case the error number represents the SQLCODE.

ADVISE_MQT table

The ADVISE_MQT table contains information about materialized query tables (MQT) recommended by the Design Advisor.

Table 94. ADVISE_MQT Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	No	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	No	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	No	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	No	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	No	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	No	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	No	Statement number within package to which this Explain information is related.
SECTNO	INTEGER	No	No	Statement number within package to which this Explain information is related.
NAME	VARCHAR(128)	No	No	MQT name.
CREATOR	VARCHAR(128)	No	No	MQT creator name.
IID	SMALLINT	No	No	Internal identifier.
CREATE_TIME	TIMESTAMP	No	No	Time at which the MQT was created.
STATS_TIME	TIMESTAMP	Yes	No	Time at which statistics were taken.
NUMROWS	DOUBLE	No	No	The number of estimated rows in the MQT.
NUMCOLS	SMALLINT	No	No	Number of columns defined in the MQT.
ROWSIZE	DOUBLE	No	No	Average length (in bytes) of a row in the MQT.
BENEFIT	FLOAT	No	No	Reserved for future use.
USE_MQT	CHAR(1)	Yes	No	Set to 'Y' when the MQT is recommended.
MQT_SOURCE	CHAR(1)	Yes	No	Indicates where the MQT candidate was generated. Set to 'I' if the MQT candidate is a refresh-immediate MQT, or 'D' if it can only be created as a full refresh-deferred MQT.
QUERY_TEXT	CLOB(2M)	No	No	Contains the query that defines the MQT.
CREATION_TEXT	CLOB(2M)	No	No	Contains the CREATE TABLE DDL for the MQT.
SAMPLE_TEXT	CLOB(2M)	No	No	Contains the sampling query that is used to get detailed statistics for the MQT. Only used when detailed statistics are required for the Design Advisor. The resulting sampled statistics will be shown in this table. If null, then no sampling query was created for this MQT.
COLSTATS	CLOB(2M)	No	No	Contains the column statistics for the MQT (if not null). These statistics are in XML format and include the column name, column cardinality and, optionally, the HIGH2KEY and LOW2KEY values.
EXTRA_INFO	BLOB(2M)	No	No	Reserved for miscellaneous output.
TBSPACE	VARCHAR(128)	No	No	The table space that is recommended for the MQT.
RUN_ID	TIMESTAMP	Yes	FK	A value corresponding to the START_TIME of a row in the ADVISE_INSTANCE table, linking it to the same Design Advisor run.
REFRESH_TYPE	CHAR(1)	No	No	Set to 'I' for immediate or 'D' for deferred.
EXISTS	CHAR(1)	No	No	Set to 'Y' if the MQT exists in the database catalog.

ADVISE_PARTITION table

The ADVISE_PARTITION table contains information about database partitions recommended by the Design Advisor, and can only be populated in a partitioned database environment.

Table 95. ADVISE_PARTITION Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
EXPLAIN_REQUESTER	VARCHAR(128)	No	No	Authorization ID of initiator of this Explain request.
EXPLAIN_TIME	TIMESTAMP	No	No	Time of initiation for Explain request.
SOURCE_NAME	VARCHAR(128)	No	No	Name of the package running when the dynamic statement was explained or name of the source file when the static SQL was explained.
SOURCE_SCHEMA	VARCHAR(128)	No	No	Schema, or qualifier, of source of Explain request.
SOURCE_VERSION	VARCHAR(64)	No	No	Version of the source of the Explain request.
EXPLAIN_LEVEL	CHAR(1)	No	No	Level of Explain information for which this row is relevant.
STMTNO	INTEGER	No	No	Statement number within package to which this Explain information is related.
SECTNO	INTEGER	No	No	Statement number within package to which this Explain information is related.
QUERYNO	INTEGER	No	No	Numeric identifier for explained SQL statement. For dynamic SQL statements (excluding the EXPLAIN SQL statement) issued through CLP or CLI, the default value is a sequentially incremented value. Otherwise, the default value is the value of STMTNO for static SQL statements and 1 for dynamic SQL statements.
QUERYTAG	CHAR(20)	No	No	Identifier tag for each explained SQL statement. For dynamic SQL statements issued through CLP (excluding the EXPLAIN SQL statement), the default value is 'CLP'. For dynamic SQL statements issued through CLI (excluding the EXPLAIN SQL statement), the default value is 'CLI'. Otherwise, the default value used is blanks.
TBNAME	VARCHAR(128)	Yes	No	Specifies the table name.
TBCREATOR	VARCHAR(128)	Yes	No	Specifies the table creator name.
PMID	SMALLINT	Yes	No	Specifies the distribution map ID.
TBSPACE	VARCHAR(128)	Yes	No	Specifies the table space in which the table resides.
COLNAMES	CLOB(2M)	Yes	No	Specifies database partition column names, separated by commas.
COLCOUNT	SMALLINT	Yes	No	Specifies the number of database partitioning columns.
REPLICATE	CHAR(1)	Yes	No	Specifies whether or not the database partition is replicated.
COST	DOUBLE	Yes	No	Specifies the cost of using the database partition.
USEIT	CHAR(1)	Yes	No	Specifies whether or not the database partition is used in EVALUATE PARTITION mode. A database partition is used if USEIT is set to 'Y' or 'y'.
RUN_ID	TIMESTAMP	Yes	FK	A value corresponding to the START_TIME of a row in the ADVISE_INSTANCE table, linking it to the same Design Advisor run.

ADVISE_TABLE table

The ADVISE_TABLE table stores the data definition language (DDL) for table creation, using the final Design Advisor recommendations for materialized query tables (MQTs), multidimensional clustered tables (MDCs), and database partitioning.

Table 96. ADVISE_TABLE Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
RUN_ID	TIMESTAMP	Yes	FK	A value corresponding to the START_TIME of a row in the ADVISE_INSTANCE table, linking it to the same Design Advisor run.
TABLE_NAME	VARCHAR(128)	No	No	Name of the table.
TABLE_SCHEMA	VARCHAR(128)	No	No	Name of the table creator.
TABLESPACE	VARCHAR(128)	No	No	The table space in which the table is to be created.
SELECTION_FLAG	VARCHAR(4)	No	No	Indicates the recommendation type. Valid values are 'M' for MQT, 'P' for database partitioning, and 'C' for MDC. This field can include any subset of these values. For example, 'MC' indicates that the table is recommended as an MQT and an MDC table.
TABLE_EXISTS	CHAR(1)	No	No	Set to 'Y' if the table exists in the database catalog.
USE_TABLE	CHAR(1)	No	No	Set to 'Y' if the table has recommendations from the Design Advisor.
GEN_COLUMNS	CLOB(2M)	No	No	Contains a generated columns string if this row includes an MDC recommendation that requires generated columns in the create table DDL.
ORGANIZE_BY	CLOB(2M)	No	No	For MDC recommendations, contains the ORGANIZE BY clause of the create table DDL.
CREATION_TEXT	CLOB(2M)	No	No	Contains the create table DDL.
ALTER_COMMAND	CLOB(2M)	No	No	Contains an ALTER TABLE statement for the table.

ADVISE_WORKLOAD table

The ADVISE_WORKLOAD table represents the statement that makes up the workload.

Table 97. ADVISE_WORKLOAD Table. PK means that the column is part of a primary key; FK means that the column is part of a foreign key.

Column Name	Data Type	Nullable?	Key?	Description
WORKLOAD_NAME	CHAR(128)	No	No	Name of the collection of SQL statements (workload) to which this statement belongs.
STATEMENT_NO	INTEGER	No	No	Statement number within the workload to which this explain information is related.
STATEMENT_TEXT	CLOB(1M)	No	No	Content of the SQL statement.
STATEMENT_TAG	VARCHAR(256)	No	No	Identifier tag for each explained SQL statement.
FREQUENCY	INTEGER	No	No	The number of times this statement appears within the workload.
IMPORTANCE	DOUBLE	No	No	Importance of the statement.
WEIGHT	DOUBLE	No	No	Priority of the statement.
COST_BEFORE	DOUBLE	Yes	No	The cost of the query (in timerons) if the recommendations are not created.
COST_AFTER	DOUBLE	Yes	No	The cost of the query (in timerons) if the recommendations are created. COST_AFTER reflects all recommendations except those that pertain to clustered indexes and multidimensional clustering (MDC).
COMPILABLE	CHAR(17)	Yes	No	Indicates any query compile errors that occured while trying to prepare the statement. If this column is NULL or does not start with SQLCA, the SQL query could be compiled by db2advis. If a compile error is found by db2advis or the Design Advisor, the COMPILABLE column value consists of an 8 byte long SQLCA.sqlcaid field, followed by a colon (:) and an 8 byte long SQLCA.sqlstate field, which is the return code for the SQL statement.

Appendix D. Explain operators

CMPEXP operator

Operator name: CMPEXP

Represents: The computation of expressions required for intermediate or final results.

(This operator is for debug mode only.)

Related concepts:

- "Operator" in *Administration Guide: Implementation*

DELETE operator

Operator name: DELETE

Represents: The deletion of rows from a table.

This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be deleted.

Performance Suggestion:

- If you are deleting all rows from a table, consider using the DROP TABLE statement or the **LOAD REPLACE** command.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

EISCAN operator

Operator name: EISCAN

Represents: This operator scans a user defined index to produce a reduced stream of rows. The scanning uses the multiple start/stop conditions from the user supplied range producer function.

This operation is performed to narrow down the set of qualifying rows before accessing the base table (based on predicates).

Performance Suggestion:

- Over time, database updates may cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- If statistics are not current, update them using the runstats command.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

FETCH operator

Operator name: FETCH

Represents: The fetching of columns from a table using a specific row identifier (RID).

Performance suggestions:

- Expand index keys to include the fetched columns so that the data pages do not have to be accessed.
- Find the index related to the fetch, and double-click on its node to display its statistics window. Ensure that the degree of clustering is high for the index.
- Increase the buffer size if the input/output (I/O) incurred by the fetch is greater than the number of pages in the table.
- If statistics are not current, update them.

  The quantile and frequent value statistics provide information on the selectivity of predicates, which determines when index scans are chosen over table scans. To update these statistics, use the **runstats** command on a table with the WITH DISTRIBUTION clause.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

FILTER operator

Operator name: FILTER

Represents: The application of residual predicates so that data is filtered based on the criteria supplied by the predicates.

Performance suggestions:

- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the selectivity value for the predicates represents the portion of the table that you want returned.
- Ensure that the optimization class is at least 3 so that the optimizer uses a join instead of a subquery. If this is not possible, try rewriting the SQL query by hand to eliminate the subquery.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

GENROW operator

Operator name: GENROW

Represents: A built-in function that generates a table of rows, using no input from tables, indexes, or operators.

GENROW may be used by the optimizer to generate rows of data (for example, for an INSERT statement or for some IN-lists that are transformed into joins).

To view the estimated statistics for the tables generated by the GENROW function, double-click on its node.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

GRPBY operator

Operator name: GRPBY

Represents: The grouping of rows according to common values of designated columns or functions. This operation is required to produce a group of values, or to evaluate set functions.

If no GROUP BY columns are specified, the GRPBY operator may still be used if there are aggregation functions in the SELECT list, indicating that the entire table is treated as a single group when doing that aggregation.

Performance suggestions:

- This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be grouped.
- To improve the performance of a SELECT statement that contains a single aggregate function but no GROUP BY clause, try the following:
  - For a MIN(C) aggregate function, create an ascending index on C.
  - For a MAX(C) aggregate function, create a descending index on C.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

HSJOIN operator

Operator name: HSJOIN

Represents: A hash join for which the qualified rows from tables are hashed to allow direct joining, without pre-ordering the content of the tables.

A join is necessary whenever there is more than one table referenced in a FROM clause. A hash join is possible whenever there is a join predicate that equates columns from two different tables. The join predicates need to be exactly the same data type. Hash joins may also arise from a rewritten subquery, as is the case with NLJOIN .

A hash join does not require the input tables be ordered. The join is performed by scanning the inner table of the hash join and generating a lookup table by hashing the join column values. It then reads the outer table, hashing the join column values, and checking in the lookup table generated for the inner table.

Performance suggestions:

- Use local predicates (that is, predicates that reference one table) to reduce the number of rows to be joined.
- Increase the size of the sort heap to make it large enough to hold the hash lookup table in memory.

- If statistics are not current, update them using the **RUNSTATS** command.

Related concepts:
- "Operator" in *Administration Guide: Implementation*

INSERT operator

Operator name: INSERT

Represents: The insertion of rows into a table.

This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be inserted.

Related concepts:
- "Operator" in *Administration Guide: Implementation*

IXAND operator

Operator name: IXAND

Represents: The ANDing of the results of multiple index scans using Dynamic Bitmap techniques. The operator allows ANDed predicates to be applied to multiple indexes, in order to reduce underlying table accesses to a minimum.

This operator is performed to:
- Narrow down the set of rows before accessing the base table
- AND together predicates applied to multiple indexes
- AND together the results of semijoins, used in star joins.

Performance suggestions:
- Over time, database updates may cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- If statistics are not current, update them using the runstats command .
- In general, index scans are most effective when only a few rows qualify. To estimate the number of qualifying rows, the optimizer uses the statistics that are available for the columns referenced in predicates. If some values occur more frequently than others, it is important to request distribution statistics by using the WITH DISTRIBUTION clause for the **RUNSTATS command**. By using the non-uniform distribution statistics, the optimizer can distinguish among frequently and infrequently occurring values.
- IXAND can best exploit single column indexes, as start and stop keys are critical in the use of IXAND.
- For star joins , create single-column indexes for each of the most selective columns in the fact table and the related dimension tables.

Related concepts:
- "Operator" in *Administration Guide: Implementation*

IXSCAN operator

Operator name: IXSCAN

Represents: The scanning of an index to produce a reduced stream of row IDs. The scanning can use optional start/stop conditions, or may apply to indexable predicates that reference columns of the index.

This operation is performed to narrow down the set of qualifying row IDs before accessing the base table (based on predicates).

Performance suggestions:

- Over time, database updates may cause an index to become fragmented, resulting in more index pages than necessary. This can be corrected by dropping and recreating the index, or reorganizing the index.
- When two or more tables are being accessed, access to the inner table via an index may be made more efficient by providing an index on the join column of the outer table.

  For more guidelines about indexes, see the online help for Visual Explain.
- If statistics are not current, update them using the **RUNSTATS** command.
- In general, index scans are most effective when only a few row IDs qualify. To estimate the number of qualifying row IDs, the optimizer uses the statistics that are available for the columns referenced in predicates. If some values occur more frequently than others, it is important to request distribution statistics by using the WITH DISTRIBUTION clause for the **runstats** command. By using the non-uniform distribution statistics, the optimizer can distinguish among frequently and infrequently occurring values.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

MSJOIN operator

Operator name: MSJOIN

Represents: A merge join for which the qualified rows from both outer and inner tables must be in join-predicate order. A merge join is also called a *merge scan join* or a *sorted merge join* .

A join is necessary whenever there is more than one table referenced in a FROM clause. A merge join is possible whenever there is a join predicate that equates columns from two different tables. It may also arise from a rewritten subquery.

A merge join requires ordered input on joining columns, since the tables are typically scanned only once. This ordered input is obtained by accessing an index or a sorted table.

Performance suggestions:

- Use local predicates (that is, predicates that reference one table) to reduce the number of rows to be joined.
- If statistics are not current, update them using the **RUNSTATS** command.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

NLJOIN operator

Operator name: NLJOIN

Represents: A nested loop join that scans (usually with an index scan) the inner table once for each row of the outer table.

A join is necessary whenever there is more than one table referenced in a FROM clause. A nested loop join does not require a join predicate, but generally performs better with one.

A nested loop join is performed either:
- By scanning through the inner table for each accessed row of the outer table.
- By performing an index lookup on the inner table for each accessed row of the outer table.

Performance suggestions:
- A nested loop join is likely to be more efficient if there is an index on the join-predicate columns of the inner table (the table displayed to the right of the NLJOIN operator). Check to see if the inner table is a TBSCAN rather than an IXSCAN. If it is, consider adding an index on its join columns.

  Another (less important) way to make the join more efficient is to create an index on the join columns of the outer table so that the outer table is ordered.
- If statistics are not current, update them using the **RUNSTATS** command.

Related concepts:
- "Operator" in *Administration Guide: Implementation*

PIPE operator

Operator name: PIPE

Represents: The transfer of rows to other operators without any change to the rows.

(This operator is for debug mode only.)

Related concepts:
- "Operator" in *Administration Guide: Implementation*

RETURN operator

Operator name: RETURN

Represents: The return of data from a query to the user. This is the final operator in the access plan graph and shows the total accumulated values and costs for the access plan.

This operator represents a necessary operation.

Performance Suggestion:

- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the selectivity value for the predicates represents the portion of the table that you want returned.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

RIDSCN operator

Operator name: RIDSCN

Represents: The scan of a list of row identifiers (RIDs) obtained from one or more indexes.

This operator is considered by the optimizer when:

- Predicates are connected by OR keywords, or there is an IN predicate. A technique called index ORing can be used, which combines results from multiple index accesses on the same table.
- It is beneficial to use list prefetch for a single index access, since sorting the row identifiers before accessing the base rows makes the I/O more efficient.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

RPD operator

Operator name: RPD

Represents: An operator used in the federated system to retrieve data from a remote data source via a non-relational wrapper.

This operator is considered by the optimizer when it contains a remote plan that will not be inspected by the optimizer. An RPD operator sends a request to a remote non-relational data source to retrieve the query result. The request is generated by the non-relational wrapper using the API supported by the data source.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

SHIP operator

Operator name: SHIP

Represents: An operator used in the federated system to retrieve data from a remote data source. This operator is considered by the optimizer when it contains a remote plan that will not be inspected by the optimizer. A SHIP operator sends an SQL or XQuery SELECT statement to a remote data source to retrieve the query result. The SELECT statement is generated using the SQL or XQuery dialect supported by the data source, and can contain any valid query as allowed by the data source.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

SORT operator

Operator name: SORT

Represents: The sorting of the rows in a table into the order of one or more of its columns, optionally eliminating duplicate entries.

Sorting is required when no index exists that satisfies the requested ordering, or when sorting would be less expensive than an index scan. Sorting is usually performed as a final operation once the required rows are fetched, or to sort data prior to a join or a group by.

If the number of rows is high or if the sorted data cannot be piped, the operation requires the costly generation of temporary tables.

Performance suggestions:

- Consider adding an index on the sort columns.
- Ensure that you have used predicates that retrieve only the data you need. For example, ensure that the selectivity value for the predicates represents the portion of the table that you want returned.
- Check that the prefetch size of the system temporary table space is adequate, that is, it is not I/O bound. (To check this, select **Statement–>Show statistics–>Table spaces** .)
- If frequent large sorts are required, consider increasing the values of the following configuration parameters:
  - Sort heap size (sortheap). To change this parameter, right-click on the database in the Control Center, and then select **Configure** from its pop-up menu. Select the Performance tab from the notebook that appears.
  - Sort heap threshold (sheapthres). To change this parameter, right-click on the database instance in the Control Center, and then select **Configure** from its pop-up menu. Select the Performance tab from the notebook that appears.
- If statistics are not current, update them using the **RUNSTATS** command.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

TBSCAN operator

Operator name: TBSCAN

Represents: A table scan (relation scan) that retrieves rows by reading all the required data directly from the data pages.

This type of scan is chosen by the optimizer over an index scan when:

- The range of values scanned occurs frequently (that is, most of the table must be accessed)
- The table is small
- Index clustering is low
- An index does not exist

Performance suggestions:

- An index scan is more efficient than a table scan if the table is large, with most of the table's rows not being accessed. To increase the possibility that an index scan will be used by the optimizer for this situation, consider adding indexes on columns for which there are selective predicates.
- If an index already exists but was not used, check that there are selective predicates on each of its leading columns. If these predicates do exist, next check that the degree of clustering is high for the index. (To see this statistic, open the Table Statistics window for the table beneath the sort, and select its *Indexes* push button to bring up the the Index Statistics window.)
- Check that the prefetch size of the table space is adequate that is, it is not I/O bound. (To check this, select **Statement–>Show statistics–>Table spaces**.)
- If the statistics are not current, update them using the **RUNSTATS** command.

  The quantile and frequent value statistics provide information on the selectivity of predicates. For example, these statistics would be used to determine when index scans are chosen over table scans. To update these values, use the **RUNSTATS** command on a table with the WITH DISTRIBUTION clause.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

TEMP operator

Operator name: TEMP

Represents: The action of storing data in a temporary table, to be read back out by another operator (possibly multiple times). The table is removed after the SQL or XQuery statement is processed, if not before.

This operator is required to evaluate subqueries or to store intermediate results. In some situations (such as when the statement can be updated), it may be mandatory.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

TQUEUE operator

Operator name: TQUEUE

Represents: A table queue that is used to pass table data from one database agent to another when there are multiple database agents processing a query. Multiple database agents are used to process a query when parallelism is involved.

Table queue types are:

- **Local:** The table queue is used to pass data between database agents within a single node. A local table queue is used for intra-partition parallelism.
- **Non-Local:** The table queue is used to pass data between database agents on different nodes.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

UNION operator

Operator name: UNION

Represents: The concatenation of streams of rows from multiple tables.

This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be concatenated.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

UNIQUE operator

Operator name: UNIQUE

Represents: The elimination of rows having duplicate values for specified columns.

Performance Suggestion:

- This operator is not necessary only if a unique index exists on appropriate columns.

  For guidelines about indexes, see Creating appropriate indexes in the online help for Visual Explain.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

UPDATE operator

Operator name: UPDATE

Represents: The updating of data in the rows of a table.

This operator represents a necessary operation. To improve access plan costs, concentrate on other operators (such as scans and joins) that define the set of rows to be updated.

Related concepts:

- "Operator" in *Administration Guide: Implementation*

Appendix E. SQL and XQuery explain tools

You can use the db2expln and dynexpln tools to understand the access plan chosen for a particular query statement. You can also use the integrated Explain Facility in the Control Center in conjunction with Visual Explain to understand the access plan chosen for a particular query statement. Both dynamic and static query statements can be explained using the Explain Facility. One difference from the Explain tools is that with Visual Explain the Explain information is presented in a graphical format. Otherwise the level of detail provided in the two methods is equivalent.

To fully use the output of `db2expln`, and `dynexpln` you must understand:

- The different query statements supported and the terminology related to those statements (such as predicates in a SELECT statement)
- The purpose of a package (access plan)
- The purpose and contents of the system catalog tables
- General application tuning concepts

The topics in this section provide information about `db2expln` and `dynexpln`.

SQL and XQuery Explain tools

The `db2expln` tool describes the access plan selected for SQL and XQuery statements. It can be used to obtain a quick explanation of the chosen access plan when explain data was not captured. For static SQL and XQuery statements, `db2expln` examines the packages stored in the system catalog tables. For dynamic SQL and XQuery statements, `db2expln` examines the sections in the query cache.

The `dynexpln` tool can also be used to describe the access plan selected for dynamic statements. It creates a static package for the statements and then uses the `db2expln` tool to describe them. However, because the dynamic SQL and XQuery statements can be examined by `db2expln`, this utility is retained only for backward compatibility.

The explain tools (`db2expln` and `dynexpln`) are located in the `bin` subdirectory of your instance `sqllib` directory. If `db2expln` and `dynexpln` are not in your current directory, they must be in a directory that appears in your PATH environment variable.

The `db2expln` program connects and uses the `db2expln.bnd`, `db2exsrv.bnd`, and `db2exdyn.bnd` files to bind itself to a database the first time the database is accessed.

To run `db2expln`, you must have the SELECT privilege on the system catalog views as well as the EXECUTE privilege for the `db2expln`, `db2exsrv`, and `db2exdyn` packages. To run `dynexpln`, you must have BINDADD authority for the database, and the SQL schema you are using to connect to the database must exist or you must have the IMPLICIT_SCHEMA authority for the database. To explain dynamic SQL and XQuery statements using either `db2expln` or `dynexpln`, you must also

have any privileges needed for the query statements being explained. (Note that if you have SYSADM or DBADM authority, you will automatically have all these authorization levels.)

Related concepts:

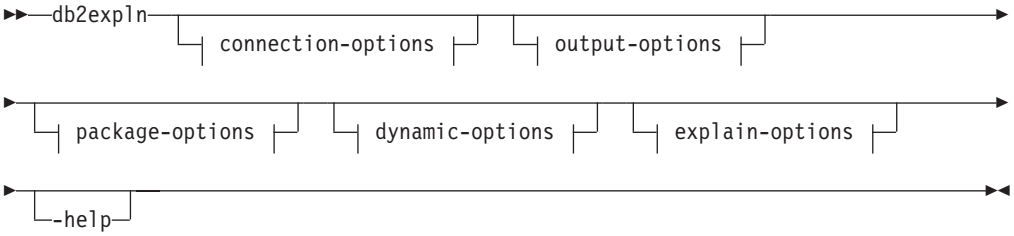
- "Guidelines for analyzing explain information" on page 233
- "Guidelines for capturing explain information" on page 231
- "Explain facility" on page 221

db2expln

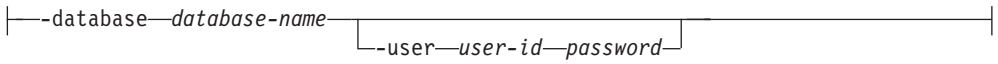
The following sections describe the syntax and parameters for *db2expln* and provide usage notes.

db2expln - SQL and XQuery Explain

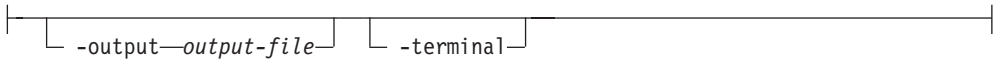
Command syntax:



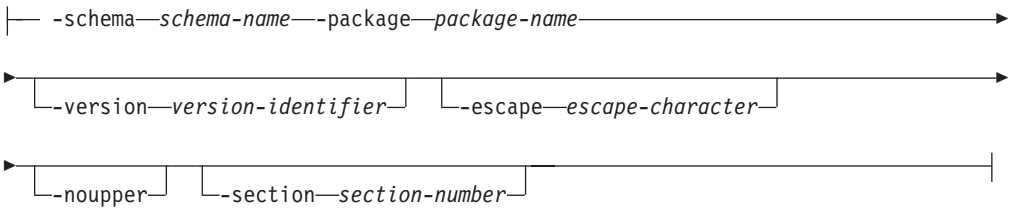
connection-options:



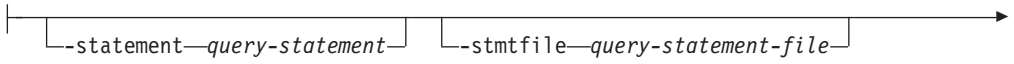
output-options:



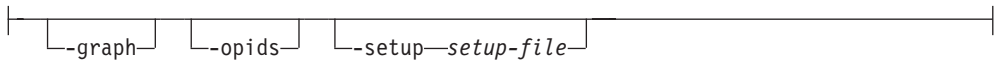
package-options:



dynamic-options:



explain-options:



Command parameters:
The options can be specified in any order.

connection-options:

These options specify the database to connect to and any options necessary to make the connection. The connection options are required except when the **-help** option is specified.

-database *database-name*
The name of the database that contains the packages to be explained.

For backward compatibility, you can use **-d** instead of **-database**.

-user *user-id password*
The authorization ID and password to use when establishing the database connection. Both *user-id* and *password* must be valid according to DB2 naming conventions and must be recognized by the database.

For backward compatibility, you can use **-u** instead of **-user**.

output-options:

These options specify where the `db2expln` output should be directed. Except when the **-help** option is specified, you must specify at least one output option. If you specify both options, output is sent to a file as well as to the terminal.

-output *output-file*
The output of `db2expln` is written to the file that you specify.

For backward compatibility, you can use **-o** instead of **-output**.

-terminal
The `db2expln` output is directed to the terminal.

For backward compatibility, you can use **-t** instead of **-terminal**.

package-options:

These options specify one or more packages and sections to be explained. Only static queries in the packages and sections are explained.

As in a LIKE predicate, you can use the pattern matching characters, which are percent sign (%) and underscore (_), to specify the *schema-name*, *package-name*, and *version-identifier*.

-schema *schema-name*
The SQL schema of the package or packages to be explained.

For backward compatibility, you can use **-c** instead of **-schema**.

-package *package-name*
The name of the package or packages to be explained.

For backward compatibility, you can use **-p** instead of **-package**.

-version *version-identifier*
The version identifier of the package or packages to be explained. The default version is the empty string.

-escape *escape-character*
The character, *escape-character* to be used as the escape character for pattern matching in the *schema-name*, *package-name*, and *version-identifier*.

For example, the `db2expln` command to explain the package TESTID.CALC% is as follows:

```
db2expln -schema TESTID -package CALC% ....
```

However, this command would also explain any other plans that start with CALC. To explain only the TESTID.CALC% package, you must use an escape character. If you specify the exclamation point (!) as the escape character, you can change the command to read: `db2expln -schema TESTID -escape ! -package CALC!% ...` . Then the ! character is used as an escape character and thus !% is interpreted as the % character and not as the "match anything" pattern. There is no default escape character.

For backward compatibility, you can use **-e** instead of **-escape**.

To avoid problems, do not specify the operating system escape character as the **db2expln** escape character.

-noupper
Specifies that the *schema-name*, *package-name*, and *version-identifier*, should not be converted to upper case before searching for matching packages.

By default, these variables are converted to upper case before searching for packages. This option indicates that these values should be used exactly as typed.

For backward compatibility, you can use **-l**, which is a lowercase L and not the number 1, instead of **-noupper**.

-section *section-number*
The section number to explain within the selected package or packages.

To explain all the sections in each package, use the number zero (0). This is the default behavior. If you do not specify this option, or if *schema-name*, *package-name*, or *version-identifier* contain a pattern-matching character, all sections are displayed.

To find section numbers, query the system catalog view SYSCAT.STATEMENTS. Refer to the *SQL Reference* for a description of the system catalog views.

For backward compatibility, you can use **-s** instead of **-section**.

dynamic-options:

These options specify one or more dynamic query statements to be explained.

-statement *query-statement*
An SQL or XQuery query statement to be dynamically prepared and explained. To explain more than one statement, either use the **-stmtfile** option to provide a file containing the query statements to explain, or use

the **-terminator** option to define a termination character that can be used to separate statements in the **-statement** option.

For compatibility with `dynexpln`, you can use **-q** instead of **-statement**.

-stmtfile *query-statement-file*
A file that contains one or more query statements to be dynamically prepared and explained. By default, each line of the file is assumed to be a distinct query statement. If statements must span lines, use the **-terminator** option to specify the character that marks the end of an query statement.

For compatibility with `dynexpln`, you can use **-f** instead of **-stmtfile**.

-terminator *termination-character*
The character that indicates the end of dynamic query statements. By default, the **-statement** option provides a single query statement and each line of the file in the **-stmtfile** is treated as a separate query statement. The termination character that you specify can be used to provide multiple query statements with **-statement** or to have statements span lines in the **-stmtfile** file.

For compatibility with `dynexpln`, you can use **-z** instead of **-terminator**.

-noenv
Specifies that dynamic statements that alter the compilation environment should not be executed after they have been explained.

By default, db2expln will execute any of the following statements after they have been explained:

```
SET CURRENT DEFAULT TRANSFORM GROUP
SET CURRENT DEGREE
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
SET CURRENT QUERY OPTIMIZATION
SET CURRENT REFRESH AGE
SET PATH
SET SCHEMA
```

These statements make it possible to alter the plan chosen for subsequent dynamic query statements processed by `db2expln`.

If you specify **-noenv**, then these statement are explained, but not executed.

It is necessary to specify either **-statement** or **-stmtfile** to explain dynamic query. Both options can be specified in a single invocation of `db2expln`.

explain-options:

These options determine what additional information is provided in the explained plans.

-graph Show optimizer plan graphs. Each section is examined, and the original optimizer plan graph is constructed as presented by Visual Explain.

The generated graph may not match the Visual Explain graph exactly. It is possible for the optimizer graph to show some gaps, based on the information contained within the section plan.

For backward compatibility, you can specify **-g** instead of **-graph**.

-opids Display operator ID numbers in the explained plan.

The operator ID numbers allow the output from db2expln to be matched to the output from the explain facility. Not all operators have an ID number and that some ID numbers that appear in the explain facility output do not appear in the db2expln output.

For backward compatibility, you can specify **-i** instead of **-opids**.

-help Shows the help text for db2expln. If this option is specified no packages are explained.

Most of the command line is processed in the *db2exsrv* stored procedure. To get help on all the available options, it is necessary to provide **connection-options** along with **-help**. For example, use:

```
db2expln -help -database SAMPLE
```

For backward compatibility, you can specify **-h** or **-?**.

-setup *setup-file*
A file that contains one or more statements needed to setup the environment for dynamic statements or for static statements that need to be recompiled (such as a static statement that references a declared global temporary table). Each statement in the file will be executed and any errors or warnings will be reported. The statements in the file are not explained.

Usage notes:

Unless you specify the **-help** option, you must specify either package-options or dynamic-options. You can explain both packages and dynamic SQL with a single invocation of db2expln.

Some of the option flags above might have special meaning to your operating system and, as a result, might not be interpreted correctly in the db2expln command line. However, you might be able to enter these characters by preceding them with an operating system escape character. For more information, see your operating system documentation. Make sure that you do not inadvertently specify the operating system escape character as the db2expln escape character.

Help and initial status messages, produced by db2expln, are written to standard output. All prompts and other status messages produced by the explain tool are written to standard error. Explain text is written to standard output or to a file depending on the output option chosen.

Examples:

To explain multiple plans with one invocation of db2expln, use the **-package**, **-schema**, and **-version** option and specify string constants for packages and creators with LIKE patterns. That is, the underscore (_) can be used to represent a single character, and the percent sign (%) can be used to represent the occurrence of zero or more characters.

To explain all sections for all packages in a database named SAMPLE, with the results being written to the file **my.exp** , enter

```
db2expln -database SAMPLE -schema % -package %  -output my.exp
```

As another example, suppose a user has a CLP script file called ″statements.db2″ and wants to explain the statements in the file. The file contains the following statements:

```
SET PATH=SYSIBM, SYSFUN, DEPT01, DEPT93@
SELECT EMPNO, TITLE(JOBID) FROM EMPLOYEE@
```

To explain these statements, enter the following command:

```
db2expln -database DEPTDATA -stmtfile statements.db2 -terminator @ -terminal
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652
- "SQL and XQuery Explain tools" on page 627

Usage notes for db2expln

The following are common messages displayed by `db2expln`:

- `No packages found for database package pattern: "<creator>".<package> with version "<version>"`

  This message will appear in the output if no packages were found in the database that matched the specified pattern.
- `Bind messages can be found in db2expln.msg`

  This message will appear in the output if the bind of `db2expln.bnd` was not successful. Further information on the problems encountered will be found in the file `db2expln.msg` in the current directory.
- `Section number overridden to 0 (all sections) for potential multiple packages.`

  This message will appear in the output if multiple packages may be encountered by `db2expln`. This action will be taken if one of the pattern matching characters is used in the package or creator input arguments.
- Bind messages for <bind file> can be found in <message file>

  This message will appear if the bind of the given bind file was not successful. Further information on the problems encountered will be found in the given message file on the database server.
- `No static sections qualify from package.`

  This message will appear in the output if the specified package only contains dynamic query statements which means that there are no static sections.
- `Package "<creator>"."<package>", "<version>", is not valid. Rebind the package and then rerun db2expln.`

  This message will appear in the output if the package specified is currently not valid. As directed, reissue the BIND or REBIND command for the plan to re-create a valid package in the database, and then rerun `db2expln`.

SQL Statements Excluded: The following statements will not be explained:

- BEGIN/END DECLARE SECTION
- BEGIN/END COMPOUND
- INCLUDE
- WHENEVER
- COMMIT and ROLLBACK
- CONNECT
- OPEN cursor
- FETCH
- CLOSE cursor

- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- DESCRIBE
- Dynamic DECLARE CURSOR
- SQL control statements

Each sub-statement within a compound SQL statement may have its own section, which can be explained by **db2expln**.

Note: `db2expln` does not exclude any XQuery statements.

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Related reference:

- "db2expln - SQL and XQuery Explain" on page 628

Explain output information

The following sections describe the kind of information that *db2expln* and *dynexpln* can provide.

Description of db2expln and dynexpln output

In the output, the explain information for each package appears in the following two parts:

- Package information such as date of bind and relevant bind options
- Section information such as the section number, followed by the SQL or XQuery statement being explained. Below the section information, the explain output of the access plan chosen for the SQL or XQuery statement appears.

The steps of an access plan, or section, are presented in the order that the database manager executes them. Each major step is shown as a left-justified heading with information about that step indented below it. Indentation bars appear in the left margin of the explain output for the access plan. These bars also mark the scope of the operation. Operations at a lower level of indentation, farther to the right, in the same operation are processed before returning to the previous level of indentation.

Remember that the access plan chosen was based on an augmented version of the original SQL or XQuery statement that is shown in the output. For example, the original statement may cause triggers and constraints to be activated. In addition, the query rewrite component of the query compiler might rewrite the SQL or XQuery statement to an equivalent but more efficient format. All of these factors are included in the information that the optimizer uses when it determines the most efficient plan to satisfy the statement. Thus, the access plan shown in the explain output may differ substantially from the access plan that you might expect for the original SQL or XQuery statement. The explain facility, which includes the explain tables, SET CURRENT EXPLAIN mode, and Visual Explain, shows the actual SQL or XQuery statement used for optimization in the form of an SQL- or XQuery-like statement which is created by reverse-translating the internal representation of the query.

When you compare output from `db2expln` or `dynexpln` to the output of the Explain facility, the operator ID option (**-opids**) can be very useful. Each time `db2expln` or `dynexpln` starts processing a new operator from the Explain facility, the operator ID number is printed to the left of the explained plan. The operator IDs can be used to match up the steps in the different representations of the access plan. Note that there is not always a one-to-one correspondence between the operators in the Explain facility output and the operations shown by `db2expln` and `dynexpln`.

Related concepts:

- "Aggregation information" on page 646
- "Block and row identifier preparation information" on page 645
- "Data stream information" on page 644
- "Explain facility" on page 221
- "Federated query information" on page 649
- "Insert, update, and delete information" on page 645
- "Join information" on page 642
- "Parallel processing information" on page 647
- "Table access information" on page 635
- "Visual Explain overview" in *Administration Guide: Implementation*
- "Temporary table information" on page 640

Related reference:

- "db2expln - SQL and XQuery Explain" on page 628

Table access information

This statement tells the name and type of table being accessed. It has two formats that are used:

1. Regular tables of three types:
   - Access Table Name:

     ```
     Access Table Name = schema.name  ID = ts,n
     ```

     where:
     - *schema.name* is the fully-qualified name of the table being accessed
     - *ID* is the corresponding TABLESPACEID and TABLEID from the SYSCAT.TABLES catalog for the table
   - Access Hierarchy Table Name:

     ```
     Access Hierarchy Table Name = schema.name  ID = ts,n
     ```

     where:
     - *schema.name* is the fully-qualified name of the table being accessed
     - *ID* is the corresponding TABLESPACEID and TABLEID from the SYSCAT.TABLES catalog for the table
   - Access Materialized Query Table Name:

     ```
     Access Materialized Query Table Name = schema.name  ID = ts,n
     ```

     where:
     - *schema.name* is the fully-qualified name of the table being accessed
     - *ID* is the corresponding TABLESPACEID and TABLEID from the SYSCAT.TABLES catalog for the table

2. Temporary tables of two types:
   - Access Temporary Table ID:

     ```
     Access Temp Table  ID = tn
     ```

     where:
     - *ID* is the corresponding identifier assigned by `db2expln`
   - Access Declared Global Temporary Table ID:

     ```
     Access Global Temp Table  ID = ts,tn
     ```

     where:
     - *ID* is the corresponding TABLESPACEID from the SYSCAT.TABLES catalog for the table (`ts`); and the corresponding identifier assigned by `db2expln` (`tn`)

Following the table access statement, additional statements will be provided to further describe the access. These statements will be indented under the table access statement. The possible statements are:

- Number of columns
- Block access
- Parallel scan
- Scan directive
- Row access method
- Lock intents
- Predicates
- Miscellaneous statements

Number of Columns

The following statement indicates the number of columns being used from each row of the table:

```
#Columns = n
```

Block Access

The following statement indicates that the table has one or more dimension block indexes defined on it:

```
Clustered by Dimension for Block Index Access
```

If this text is not shown, the table was created without the DIMENSION clause.

Parallel Scan

The following statement indicates that the database manager will use several subagents to read from the table in parallel:

```
Parallel Scan
```

If this text is not shown, the table will only be read from by one agent (or subagent).

Scan Direction

The following statement indicates that the database manager will read rows in a reverse order:

```
Scan Direction = Reverse
```

If this text is not shown, the scan direction is forward, which is the default.

Row Access Method

One of the following statements will be displayed, indicating how the qualifying rows in the table are being accessed:

- The `Relation Scan` statement indicates that the table is being sequentially scanned to find the qualifying rows.
  - The following statement indicates that no prefetching of data will be done:

    ```
    Relation Scan
    |  Prefetch: None
    ```

  - The following statement indicates that the optimizer has predetermined the number of pages that will be prefetched:

    ```
    Relation Scan
    |  Prefetch: n Pages
    ```

  - The following statement indicates that data should be prefetched:

    ```
    Relation Scan
    |  Prefetch: Eligible
    ```

  - The following statement indicates that the qualifying rows are being identified and accessed through an index:

    ```
    Index Scan:  Name = schema.name  ID = xx
    | Index type
    | Index Columns:
    ```

    where:

    - *schema.name* is the fully-qualified name of the index being scanned
    - *ID* is the corresponding IID column in the SYSCAT.INDEXES catalog view.
    - Index type is one of:

      ```
      Regular Index (Not Clustered)
      Regular Index (Clustered)
      Dimension Block Index
      Composite Dimension Block Index
      index over XML data
      ```

      This will be followed by one row for each column in the index. Each column in the index will be listed in one of the following forms:

      ```
      n: column_name (Ascending)
      n: column_name (Descending)
      n: column_name (Include Column)
      ```

      The following statements are provided to clarify the type of index scan:

    - The range delimiting predicates for the index are shown by:

      ```
      #Key Columns = n
      |  Start Key: xxxxx
      |  Stop Key: xxxxx
      ```

      Where xxxxx is one of:

      - `Start of Index`
      - `End of Index`
      - `Inclusive Value:` or `Exclusive  Value:`

An inclusive key value will be included in the index scan. An exclusive key value will not be included in the scan. The value for the key will be given by one of the following rows for each part of the key:

```
n: 'string'
n: nnn
n: yyyy-mm-dd
n: hh:mm:ss
n: yyyy-mm-dd hh:mm:ss.uuuuuu
n: NULL
n: ?
```

If a literal string is shown, only the first 20 characters are displayed. If the string is longer than 20 characters, this will be shown by `...` at the end of the string. Some keys cannot be determined until the section is executed. This is shown by a `?` as the value.

- `Index-Only Access`

  If all the needed columns can be obtained from the index key, this statement will appear and no table data will be accessed.

- The following statement indicates that no prefetching of index pages will be done:

  ```
  Index Prefetch: None
  ```

- The following statement indicates that index pages should be prefetched:

  ```
  Index Prefetch: Eligible
  ```

- The following statement indicates that no prefetching of data pages will be done:

  ```
  Data Prefetch: None
  ```

- The following statement indicates that data pages should be prefetched:

  ```
  Data Prefetch: Eligible
  ```

- If there are predicates that can be passed to the Index Manager to help qualify index entries, the following statement is used to show the number of predicates:

  ```
  Sargable Index Predicate(s)
  |  #Predicates = n
  ```

– If the qualifying rows are being accessed by using row IDs (RIDs) that were prepared earlier in the access plan, it will be indicated with the statement:

```
Fetch Direct Using Row IDs
```

If the table has one or more block indexes defined for it, then rows may be accessed by either block or row IDs. This is indicated by:

```
Fetch Direct Using Block or Row IOs
```

Lock Intents

For each table access, the type of lock that will be acquired at the table and row levels is shown with the following statement:

```
Lock Intents
|  Table: xxxx
|  Row  : xxxx
```

Possible values for a table lock are:

- Exclusive
- Intent Exclusive
- Intent None

- Intent Share
- Share
- Share Intent Exclusive
- Super Exclusive
- Update

Possible values for a row lock are:

- Exclusive
- Next Key Exclusive (does not appear in `db2expln` output)
- None
- Share
- Next Key Share
- Update
- Next Key Weak Exclusive
- Weak Exclusive

Predicates

There are two statements that provide information about the predicates used in an access plan:

1. The following statement indicates the number of predicates that will be evaluated for each block of data retrieved from a blocked index.

   ```
   Block Predicates(s)
   |  #Predicates = n
   ```

2. The following statement indicates the number of predicates that will be evaluated while the data is being accessed. The count of predicates does not include push-down operations such as aggregation or sort.

   ```
   Sargable Predicate(s)
   |  #Predicates = n
   ```

3. The following statement indicates the number of predicates that will be evaluated once the data has been returned:

   ```
   Residual Predicate(s)
   |  #Predicates = n
   ```

The number of predicates shown in the above statements may not reflect the number of predicates provided in the query statement because predicates can be:

- Applied more than once within the same query
- Transformed and extended with the addition of implicit predicates during the query optimization process
- Transformed and condensed into fewer predicates during the query optimization process.

Miscellaneous Table Statements

- The following statement indicates that only one row will be accessed:

  ```
  Single Record
  ```

- The following statement appears when the isolation level used for this table access uses a different isolation level than the statement:

  ```
  Isolation Level: xxxx
  ```

  A different isolation level may be used for a number of reasons, including:

  - A package was bound with Repeatable Read and affects referential integrity constraints; the access of the parent table to check referential integrity constraints is downgraded to an isolation level of Cursor Stability to avoid holding unnecessary locks on this table.
  - A package bound with Uncommitted Read issues a DELETE or UPDATE statement; the table access for the actual delete is upgraded to Cursor Stability.
- The following statement indicates that some or all of the rows read from the temporary table will be cached outside the buffer pool if sufficient sortheap memory is available:

  ```
  Keep Rows In Private Memory
  ```
- If the table has the volatile cardinality attribute set, it will be indicated by:

  ```
  Volatile Cardinality
  ```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Temporary table information

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. This table only exists while the access plan is being executed. Generally, temporary tables are used when subqueries need to be evaluated early in the access plan, or when intermediate results will not fit in the available memory.

If a temporary table needs to be created, then one of two possible statements may appear. These statements indicate that a temporary table is to be created and rows inserted into it. The ID is an identifier assigned by `db2expln` for convenience when referring to the temporary table. This ID is prefixed with the letter 't' to indicate that the table is a temporary table.

- The following statement indicates an ordinary temporary table will be created:

  ```
  Insert Into Temp Table  ID = tn
  ```
- The following statement indicates an ordinary temporary table will be created by multiple subagents in parallel:

  ```
  Insert Into Shared Temp Table  ID = tn
  ```
- The following statement indicates a sorted temporary table will be created:

  ```
  Insert Into Sorted Temp Table  ID = tn
  ```
- The following statement indicates a sorted temporary table will be created by multiple subagents in parallel:

  ```
  Insert Into Sorted Shared Temp Table  ID = tn
  ```
- The following statement indicates a declared global temporary table will be created:

  ```
  Insert Into Global Temp Table  ID = ts,tn
  ```
- The following statement indicates a declared global temporary table will be created by multiple subagents in parallel:

  ```
  Insert Into Shared Global Temp Table  ID = ts,tn
  ```
- The following statement indicates a sorted declared global temporary table will be created:

  ```
  Insert Into Sorted Global Temp Table  ID = ts,tn
  ```

- The following statement indicates a sorted declared global temporary table will be created by multiple subagents in parallel:

```
Insert Into Sorted Shared Global Temp Table  ID = ts,tn
```

Each of the above statements will be followed by:

```
#Columns = n
```

which indicates how many columns are in each row being inserted into the temporary table.

Sorted Temporary Tables

Sorted temporary tables can result from such operations as:

- ORDER BY
- DISTINCT
- GROUP BY
- Merge Join
- '= ANY' subquery
- '<> ALL' subquery
- INTERSECT or EXCEPT
- UNION (without the ALL keyword)

A number of additional statements may follow the original creation statement for a sorted temporary table:

- The following statement indicates the number of key columns used in the sort:

```
#Sort Key Columns = n
```

  For each column in the sort key, one of the following lines will be displayed:

```
Key n: column_name (Ascending)
Key n: column_name (Descending)
Key n: (Ascending)
Key n: (Descending)
```

- The following statements provide estimates of the number of rows and the row size so that the optimal sort heap can be allocated at run time.

```
Sortheap Allocation Parameters:
|  #Rows     = n
|  Row Width = n
```

- If only the first rows of the sorted result are needed, the following is displayed:

```
Sort Limited To Estimated Row Count
```

- For sorts in a symmetric multiprocessor (SMP) environment, the type of sort to be performed is indicated by one of the following statements:

```
Use Partitioned Sort
Use Shared Sort
Use Replicated Sort
Use Round-Robin Sort
```

- The following statements indicate whether or not the result from the sort will be left in the sort heap:

```
Piped
```

  and

```
Not Piped
```

  If a piped sort is indicated, the database manager will keep the sorted output in memory, rather than placing the sorted result in another temporary table.

- The following statement indicates that duplicate values will be removed during the sort:

  ```
  Duplicate Elimination
  ```

- If aggregation is being performed in the sort, it will be indicated by one of the following statements:

  ```
  Partial Aggregation
  Intermediate Aggregation
  Buffered Partial Aggregation
  Buffered Intermediate Aggregation
  ```

Temporary Table Completion: After a table access that contains a push-down operation to create a temporary table (that is, a create temporary table that occurs within the scope of a table access), there will be a "completion" statement, which handles end-of-file by getting the temporary table ready to provide rows to subsequent temporary table access. One of the following lines will be displayed:

```
Temp Table Completion  ID = tn
Shared Temp Table Completion  ID = tn
Sorted Temp Table Completion  ID = tn
Sorted Shared Temp Table Completion  ID = tn
```

Table Functions

Table functions are user-defined functions (UDFs) that return data to the statement in the form of a table. Table functions are indicated by:

```
Access User Defined Table Function
|  Name = schema.funcname
|  Specific Name = specificname
|  SQL Access Level = accesslevel
|  Language = lang
|  Parameter Style = parmstyle
|  Fenced                      Not Deterministic
|  Called on NULL Input        Disallow Parallel
|  Not Federated               Not Threadsafe
```

The specific name uniquely identifies the table function invoked. The remaining rows detail the atributes of the function.

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Join information

There are three types of joins:

- Hash join
- Merge join
- Nested loop join.

When the time comes in the execution of a section for a join to be performed, one of the following statements is displayed:

```
Hash Join
Merge Join
Nested Loop Join
```

It is possible for a left outer join to be performed. A left outer join is indicated by one of the following statements:

```
Left Outer Hash Join
Left Outer Merge Join
Left Outer Nested Loop Join
```

For merge and nested loop joins, the outer table of the join will be the table referenced in the previous access statement shown in the output. The inner table of the join will be the table referenced in the access statement that is contained within the scope of the join statement. For hash joins, the access statements are reversed with the outer table contained within the scope of the join and the inner table appearing before the join.

For a hash or merge join, the following additional statements may appear:

- In some circumstances, a join simply needs to determine if any row in the inner table matches the current row in the outer. This is indicated with the statement:

  ```
  Early Out: Single Match Per Outer Row
  ```

- It is possible to apply predicates after the join has completed. The number of predicates being applied will be indicated as follows:

  ```
  Residual Predicate(s)
  |  #Predicates = n
  ```

For a hash join, the following additional statements may appear:

- The hash table is built from the inner table. If the hash table build was pushed down into a predicate on the inner table access, it is indicated by the following statement in the access of the inner table:

  ```
  Process Hash Table For Join
  ```

- While accessing the outer table, a probe table can be built to improve the performance of the join. The probe table build is indicated by the following statement in the access of the outer table:

  ```
  Process Probe Table For Hash Join
  ```

- The estimated number of bytes needed to build the hash table is represented by:

  ```
  Estimated Build Size: n
  ```

- The estimated number of bytes needed for the probe table is represented by:

  ```
  Estimated Probe Size: n
  ```

For a nested loop join, the following additional statement may appear immediately after the join statement:

```
Piped Inner
```

This statement indicates that the inner table of the join is the result of another series of operations. This is also referred to as a *composite inner*.

If a join involves more than two tables, the explain steps should be read from top to bottom. For example, suppose the explain output has the following flow:

```
Access ..... W
Join
|  Access ..... X
Join
|  Access ..... Y
Join
|  Access ..... Z
```

The steps of execution would be:

1. Take a row that qualifies from `W`.

2. Join row from W with (next) row from X and call the result P1 (for partial join result number 1).
3. Join P1 with (next) row from Y to create P2 .
4. Join P2 with (next) row from Z to obtain one complete result row.
5. If there are more rows in Z, go to step 4.
6. If there are more rows in Y, go to step 3.
7. If there are more rows in X, go to step 2.
8. If there are more rows in W, go to step 1.

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Data stream information

Within an access plan, there is often a need to control the creation and flow of data from one series of operations to another. The data stream concept allows a group of operations within an access plan to be controlled as a unit. The start of a data stream is indicated by the following statement:

```
Data Stream n
```

where n is a unique identifier assigned by db2expln for ease of reference. The end of a data stream is indicated by:

```
End of Data Stream n
```

All operations between these statements are considered part of the same data stream.

A data stream has a number of characteristics and one or more statements can follow the initial data stream statement to describe these characteristics:

- If the operation of the data stream depends on a value generated earlier in the access plan, the data stream is marked with:

  ```
  Correlated
  ```

- Similar to a sorted temporary table, the following statements indicate whether or not the results of the data stream will be kept in memory:

  ```
  Piped
  ```

  and

  ```
  Not Piped
  ```

  As was the case with temporary tables, a piped data stream may be written to disk, if insufficient memory exists at execution time. The access plan will provide for both possibilities.

- The following statement indicates that only a single record is required from this data stream:

  ```
  Single Record
  ```

When a data stream is accessed, the following statement will appear in the output:

```
Access Data Stream n
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634

- "Examples of db2expln and dynexpln output" on page 652

Insert, update, and delete information

The explain text for these SQL statements is self-explanatory. Possible statement text for these SQL operations can be:

```
Insert: Table Name = schema.name ID = ts,n
Update: Table Name = schema.name ID = ts,n
Delete: Table Name = schema.name ID = ts,n
Insert: Hierarchy Table Name = schema.name ID = ts,n
Update: Hierarchy Table Name = schema.name ID = ts,n
Delete: Hierarchy Table Name = schema.name ID = ts,n
Insert: Materialized Query Table = schema.name ID = ts,n
Update: Materialized Query Table = schema.name ID = ts,n
Delete: Materialized Query Table = schema.name ID = ts,n
Insert: Global Temporary Table ID = ts, tn
Update: Global Temporary Table ID = ts, tn
Delete: Global Temporary Table ID = ts, tn
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Block and row identifier preparation information

For some access plans, it is more efficient if the qualifying row and block identifiers (IDs) are sorted and duplicates removed (in the case of index ORing) or that a technique is used to identify IDs appearing in all indexes being accessed (in the case of index ANDing) before the actual table access is performed. There are three main uses of ID preparation as indicated by the explain statements:

- Either of the following statements indicates that Index ORing is used to prepare the list of qualifying IDs:

  ```
  Index ORing Preparation
  Block Index ORing Preparation
  ```

  Index ORing refers to the technique of making more than one index access and combining the results to include the distinct IDs that appear in any of the indexes accessed. The optimizer will consider index ORing when predicates are connected by OR keywords or there is an IN predicate. The index accesses can be on the same index or different indexes.
- Another use of ID preparation is to prepare the input data to be used during list prefetch, as indicated by either of the following:

  ```
  List Prefetch Preparation
  Block List Prefetch RID Preparation
  ```
- *Index ANDing* refers to the technique of making more than one index access and combining the results to include IDs that appear in all of the indexes accessed. Index ANDing processing is started with either of these statements:

  ```
  Index ANDing
  Block Index ANDing
  ```

  If the optimizer has estimated the size of the result set, the estimate is shown with the following statement:

  ```
  Optimizer Estimate of Set Size: n
  ```

Index ANDing filter operations process IDs and use bit filter techniques to determine the IDs which appear in every index accessed. The following statements indicate that IDs are being processed for index ANDing:

```
Index ANDing Bitmap Build Using Row IDs
Index ANDing Bitmap Probe Using Row IDs
Index ANDing Bitmap Build and Probe Using Row IDs
Block Index ANDing Bitmap Build Using Block IDs
Block Index ANDing Bitmap Build and Probe Using Block IDs
Block Index ANDing Bitmap Build and Probe Using Row IDs
Block Index ANDing Bitmap Probe Using Block IDs and Build Using Row IDs
Block Index ANDing Bitmap Probe Using Block IDs
Block Index ANDing Bitmap Probe Using Row IDs
```

If the optimizer has estimated the size of the result set for a bitmap, the estimate is shown with the following statement:

```
Optimizer Estimate of Set Size: n
```

For any type of ID preparation, if list prefetch can be performed it will be indicated with the statement:

```
Prefetch: Enabled
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Aggregation information

Aggregation is performed on those rows meeting the specified criteria, if any, provided by the SQL statement predicates. If some sort of aggregate function is to be done, one of the following statements appears:

```
Aggregation
Predicate Aggregation
Partial Aggregation
Partial Predicate Aggregation
Intermediate Aggregation
Intermediate Predicate Aggregation
Final Aggregation
Final Predicate Aggregation
```

Predicate aggregation states that the aggregation operation has been pushed-down to be processed as a predicate when the data is actually accessed.

Beneath either of the above aggregation statements will be a indication of the type of aggregate function being performed:

```
Group By
Column Function(s)
Single Record
```

The specific column function can be derived from the original SQL statement. A single record is fetched from an index to satisfy a MIN or MAX operation.

If predicate aggregation is used, then subsequent to the table access statement in which the aggregation appeared, there will be an aggregation ″completion″, which carries out any needed processing on completion of each group or on end-of-file. One of the following lines is displayed:

```
Aggregation Completion
Partial Aggregation Completion
Intermediate Aggregation Completion
Final Aggregation Completion
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Parallel processing information

Executing an SQL statement in parallel (using either intra-partition or inter-partition parallelism) requires some special operations. The operations for parallel plans are described below.

- When running an intra-partition parallel plan, portions of the plan will be executed simultaneously using several subagents. The creation of the subagents is indicated by the statement:

  ```
  Process Using n Subagents
  ```

- When running an inter-partition parallel plan, the section is broken into several subsections. Each subsection is sent to one or more nodes to be run. An important subsection is the *coordinator subsection*. The coordinator subsection is the first subsection in every plan. It gets control first and is responsible for distributing the other subsections and returning results to the calling application.

  The distribution of subsections is indicated by the statement:

  ```
  Distribute Subsection #n
  ```

  The nodes that receive a subsection can be determined in one of eight ways:

  – The following indicates that the subsection will be sent to a node within the database partition group based on the value of the columns.

    ```
    Directed by Hash
    |  #Columns = n
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

  – The following indicates that the subsection will be sent to a predetermined node. (This is frequently seen when the statement uses the NODENUMBER() function.)

    ```
    Directed by Node Number
    ```

  – The following indicates that the subsection will be sent to the node corresponding to a predetermined database partition number in the given database partition group. (This is frequently seen when the statement uses the PARTITION() function.)

    ```
    Directed by Partition Number
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

  – The following indicates that the subsection will be sent to the node that provided the current row for the application's cursor.

    ```
    Directed by Position
    ```

  – The following indicates that only one node, determined when the statement was compiled, will receive the subsection.

    ```
    Directed to Single Node
    |  Node Number = n
    ```

  – Either of the following indicates that the subsection will be executed on the coordinator node.

    ```
    Directed to Application Coordinator Node
    Directed to Local Coordinator Node
    ```

  - The following indicates that the subsection will be sent to all the nodes listed.

    ```
    Broadcast to Node List
    |  Nodes = n1, n2, n3, ...
    ```

  - The following indicates that only one node, determined as the statement is executing, will receive the subsection.

    ```
    Directed to Any Node
    ```

- Table queues are used to move data between subsections in a partitioned database environment or between subagents in a symmetric multiprocessor (SMP) environment. Table queues are described as follows:
  - The following statements indicate that data is being inserted into a table queue:

    ```
    Insert Into Synchronous Table Queue  ID = qn
    Insert Into Asynchronous Table Queue  ID = qn
    Insert Into Synchronous Local Table Queue  ID = qn
    Insert Into Asynchronous Local Table Queue  ID = qn
    ```

  - For database partition table queues, the destination for rows inserted into the table queue is described by one of the following:

    All rows are sent to the coordinator node:

    ```
    Broadcast to Coordinator Node
    ```

    All rows are sent to every database partition where the given subsection is running:

    ```
    Broadcast to All Nodes of Subsection n
    ```

    Each row is sent to a database partition based on the values in the row:

    ```
    Hash to Specific Node
    ```

    Each row is sent to a database partition that is determined while the statement is executing:

    ```
    Send to Specific Node
    ```

    Each row is sent to a randomly determined node:

    ```
    Send to Random Node
    ```

  - In some situations, a database partition table queue will have to temporarily overflow some rows to a temporary table. This possibility is identified by the statement:

    ```
    Rows Can Overflow to Temporary Table
    ```

  - After a table access that contains a push-down operation to insert rows into a table queue, there will be a "completion" statement which handles rows that could not be immediately sent. One of the following lines is displayed:

    ```
    Insert Into Synchronous Table Queue Completion  ID = qn
    Insert Into Asynchronous Table Queue Completion  ID = qn
    Insert Into Synchronous Local Table Queue Completion  ID = qn
    Insert Into Asynchronous Local Table Queue Completion  ID = qn
    ```

  - The following statements indicate that data is being retrieved from a table queue:

    ```
    Access Table Queue  ID = qn
    Access Local Table Queue  ID = qn
    ```

    These messages are always followed by an indication of the number of columns being retrieved.

    ```
    #Columns = n
    ```

– If the table queue sorts the rows at the receiving end, the table queue access will also have one of the following messages:

```
Output Sorted
Output Sorted and Unique
```

These messages are followed by an indication of the number of keys used for the sort operation.

```
#Key Columns = n
```

For each column in the sort key, one of the following is displayed:

```
Key n: (Ascending)
Key n: (Descending)
```

– If predicates will be applied to rows by the receiving end of the table queue, the following message is shown:

```
Residual Predicate(s)
|  #Predicates = n
```

- Some subsections in a partitioned database environment explicitly loop back to the start of the subsection with the statement:

```
Jump Back to Start of Subsection
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Federated query information

Executing an SQL statement in a federated database requires the ability to perform portions of the statement on other data sources.

The following indicates that a data source will be read:

```
Ship Distributed Subquery #n
|   #Columns = n
```

It is possible to apply predicates to the data returned from the distributed subquery. The number of predicates being applied will be indicated as follows:

```
Residual Predicate(s)
|   #Predicates = n
```

An insert, update, or delete operation that occurs at a data source will be indicated by the appropriate message:

```
Ship Distributed Insert #n
Ship Distributed Update #n
Ship Distributed Delete #n
```

If a table is being explicitly locked at a data source, this will be indicated with the statement:

```
Ship Distributed Lock Table #n
```

DDL statements against a data source are split into two parts. The part invoked at the data source is indicated by:

```
Ship Distributed DDL Statement #n
```

If the federated server is a partitioned database, then part of the DDL statement must be run at he catalog node. This is indicated by:

```
Distributed DDL Statement #n Completion
```

The detail for each distributed substatement is provided separately. The options for distributed statements are described below:

- The data source for the subquery is shown by one of the following:

  ```
  Server: server_name (type, version)
  Server: server_name (type)
  Server: server_name
  ```

- If the data source is relational, the SQL for the substatement is displayed as:

  ```
  SQL Statement:
       statement
  ```

  Non-relational data sources are indicated with:

  ```
  Non-Relational Data Source
  ```

- The nicknames referenced in the substatement are listed as follows:

  ```
  Nicknames Referenced:
     schema.nickname ID = n
  ```

  If the data source is relational, the base table for the nickname is shows as:

  ```
  Base = baseschema.basetable
  ```

  If the data source is non-relational, the source file for the nickname is shown as:

  ```
  Source File = filename
  ```

- If values are passed from the federated server to the data source before executing the substatement, the number of values will be shown by:

  ```
  #Input Columns: n
  ```

- If values are passed from the data source to the federated server after executing the substatement, the number of values will be shown by:

  ```
  #Output Columns: n
  ```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Guidelines for analyzing where a federated query is evaluated" on page 214

Miscellaneous explain information

- Sections for data definition language statements will be indicated in the output with the following:

  ```
  DDL Statement
  ```

  No additional explain output is provided for DDL statements.

- Sections for SET statements for the updatable special registers such as **CURRENT EXPLAIN SNAPSHOT** will be indicated in the output with the following:

  ```
  SET Statement
  ```

  No additional explain output is provided for SET statements.

- If the SQL statement contains the DISTINCT clause, the following text may appear in the output:

  ```
  Distinct Filter  #Columns = n
  ```

  where `n` is the number of columns involved in obtaining distinct rows. To retrieve distinct row values, the rows must be ordered so that duplicates can be skipped. This statement will not appear if the database manager does not have to explicitly eliminate duplicates, as in the following cases:

  - A unique index exists and all the columns in the index key are part of the DISTINCT operation
  - Duplicates that can be eliminated during sorting.
- The following statement will appear if the next operation is dependent on a specific record identifier:

  ```
  Positioned Operation
  ```

  If the position operation is against a federated data source, then the statement is:

  ```
  Distributed Positioned Operation
  ```

  This statement would appear for any SQL statement that uses the WHERE CURRENT OF syntax.
- The following statement will appear if there are predicates that must be applied to the result but that could not be applied as part of another operation:

  ```
  Residual Predicate Application
  |  #Predicates = n
  ```
- The following statement will appear if there is a UNION operator in the SQL statement:

  ```
  UNION
  ```
- The following statement will appear if there is an operation in the access plan, whose sole purpose is to produce row values for use by subsequent operations:

  ```
  Table Constructor
  |  n-Row(s)
  ```

  Table constructors can be used for transforming values in a set into a series of rows that are then passed to subsequent operations. When a table constructor is prompted for the next row, the following statement will appear:

  ```
  Access Table Constructor
  ```
- The following statement will appear if there is an operation which is only processed under certain conditions:

  ```
  Conditional Evaluation
  |  Condition #n:
  |  #Predicates = n
  |  Action #n:
  ```

  Conditional evaluation is used to implement such activities as the SQL CASE statement or internal mechanisms such as referential integrity constraints or triggers. If no action is shown, then only data manipulation operations are processed when the condition is true.
- One of the following statements will appear if an ALL, ANY, or EXISTS subquery is being processed in the access plan:
  - `ANY/ALL Subquery`
  - `EXISTS Subquery`
  - `EXISTS SINGLE Subquery`
- Prior to certain UPDATE and DELETE operations, it is necessary to establish the position of a specific row within the table. This is indicated by the following statement:

  ```
  Establish Row Position
  ```
- The following statement will appear if there are rows being returned to the application:

  ```
  Return Data to Application
  |  #Columns = n
  ```

If the operation was pushed-down into a table access, it will require a completion phase. This phase appears as:

```
Return Data Completion
```

- The following information will appear if a stored procedure is being invoked:

```
Call Stored Procedure
|   Name = schema.funcname
|   Specific Name = specificname
|   SQL Access Level = accesslevel
|   Language = lang
|   Parameter Style = parmstyle
|   Expected Result Sets = n
|   Fenced                      Not Deterministic
|   Called on NULL Input        Disallow Parallel
|   Not Federated               Not Threadsafe
```

- The following information will appear if one or more LOB locators are being freed:

```
Free LOB Locators
```

Related concepts:

- "Description of db2expln and dynexpln output" on page 634
- "Examples of db2expln and dynexpln output" on page 652

Examples of db2expln and dynexpln Output

The following output examples show the explain information collected for specific queries in specific environments.

Examples of db2expln and dynexpln output

The examples shown here can help you understand the layout and format of the output from `db2expln` and `dynexpln`. These examples were run against the SAMPLE database that is provided with DB2, unless shown otherwise. A brief discussion is included with each example. Significant differences from one example to the next have been shown in **bold**.

Related concepts:

- "dynexpln" on page 222
- "Example eight: XISCAN operator" on page 665
- "Example five: federated database plan" on page 660
- "Example four: multipartition plan with inter-partition and intra-partition parallelism" on page 658
- "Example one: no parallelism" on page 653
- "Example seven: XSCAN operator" on page 664
- "Example six: XANDOR and XISCAN operators" on page 662
- "Example three: multipartition plan with inter-partition parallelism" on page 656
- "Example two: single-partition plan with intra-partition parallelism" on page 654

Related reference:

- "db2expln - SQL and XQuery Explain" on page 628

Example one: no parallelism

This example is simply requesting a list of all employee names, their jobs, department name and location, and the project names on which they are working. The essence of this access plan is that hash joins are used to join the relevant data from each of the specified tables. Since no indexes are available, the access plan does a relation scan of each table as it is joined.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:05:00

 Bind Timestamp = 2002-01-04-14.05.00.415403

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 120.518692
Estimated Cardinality = 221.535980

(    6) Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
         |  #Columns = 3
         |  Relation Scan
         |  |  Prefetch: Eligible
         |  Lock Intents
         |  |  Table: Intent Share
         |  |  Row  : Next Key Share
(    6) |  Process Build Table for Hash Join
(    2) Hash Join
         |  Estimated Build Size: 7111
         |  Estimated Probe Size: 9457
(    5)  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
         |  |  #Columns = 2
         |  |  Relation Scan
         |  |  |  Prefetch: Eligible
         |  |  Lock Intents
         |  |  |  Table: Intent Share
         |  |  |  Row  : Next Key Share
(    5)  |  |  Process Build Table for Hash Join
(    3)  |  Hash Join
         |  |  Estimated Build Size: 5737
         |  |  Estimated Probe Size: 6421
(    4)  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
         |  |  |  #Columns = 3
         |  |  |  Relation Scan
```

```
        |  |  |  |  Prefetch: Eligible
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent Share
        |  |  |  |  Row  : Next Key Share
(    4) |  |  |  Process Probe Table for Hash Join
(    1) Return Data to Application
        |  #Columns = 5

End of section


Optimizer Plan:

               RETURN
               (   1)
                 |
               HSJOIN
               (   2)
              /      \
         HSJOIN         TBSCAN
         (   3)         (   6)
        /      \          |
    TBSCAN    TBSCAN   Table:
    (   4)    (   5)   DOOLE
      |         |      EMPLOYEE
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

The first part of the plan accesses the DEPARTMENT and PROJECT tables and uses a hash join to join them. The result of this join is joined to the EMPLOYEE table. The resulting rows are returned to the application.

Example two: single-partition plan with intra-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled for a four-way SMP machine.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:12:38

 Bind Timestamp = 2002-01-04-14.12.38.732627

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
```

```
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 133.934692
Estimated Cardinality = 221.535980

(    2) Process Using 4 Subagents
(    7) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
        |  |  #Columns = 3
        |  |  Parallel Scan
        |  |  Relation Scan
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    7) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
        |  |  Estimated Build Size: 7111
        |  |  Estimated Probe Size: 9457
(    6) |  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
        |  |  |  #Columns = 2
        |  |  |  Parallel Scan
        |  |  |  Relation Scan
        |  |  |  |  Prefetch: Eligible
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent Share
        |  |  |  |  Row  : Next Key Share
(    6) |  |  |  Process Build Table for Hash Join
(    4) |  |  Hash Join
        |  |  |  Estimated Build Size: 5737
        |  |  |  Estimated Probe Size: 6421
(    5) |  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
        |  |  |  |  #Columns = 3
        |  |  |  |  Parallel Scan
        |  |  |  |  Relation Scan
        |  |  |  |  |  Prefetch: Eligible
        |  |  |  |  Lock Intents
        |  |  |  |  |  Table: Intent Share
        |  |  |  |  |  Row  : Next Key Share
(    5) |  |  |  |  Process Probe Table for Hash Join
(    2) |  Insert Into Asynchronous Local Table Queue  ID = q1
(    2) Access Local Table Queue  ID = q1  #Columns = 5
(    1) Return Data to Application
        |  #Columns = 5

End of section


Optimizer Plan:

              RETURN
              (   1)
                |
               LTQ
              (   2)
                |
              HSJOIN
              (   3)
             /      \
         HSJOIN      TBSCAN
         (   4)      (   7)
        /      \       |
    TBSCAN    TBSCAN   Table:
    (   5)    (   6)   DOOLE
```

```
    |         |      EMPLOYEE
 Table:      Table:
 DOOLE       DOOLE
 DEPARTMENT  PROJECT
```

This plan is almost identical to the plan in the first example. The main differences are the creation of four subagents when the plan first starts and the table queue at the end of the plan to gather the results of each of subagent's work before returning them to the application.

Example three: multipartition plan with inter-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled on a partitioned database made up of three database partitions.

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:54:57

 Bind Timestamp = 2002-01-04-14.54.57.033666

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = Yes
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

---------- SECTION ----------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 118.483406
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
           |  Broadcast to Node List
           |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
           |  #Columns = 5

        Subsection #1:
(    8)    Access Table Queue  ID = q2  #Columns = 2
```

```
(    3)    Hash Join
           |  Estimated Build Size: 5737
           |  Estimated Probe Size: 8015
(    6)    |  Access Table Queue  ID = q3  #Columns = 3
(    4)    |  Hash Join
           |  |  Estimated Build Size: 5333
           |  |  Estimated Probe Size: 6421
(    5)    |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
           |  |  |  #Columns = 3
           |  |  |  Relation Scan
           |  |  |  |  Prefetch: Eligible
           |  |  |  Lock Intents
           |  |  |  |  Table: Intent Share
           |  |  |  |  Row  : Next Key Share
(    5)    |  |  |  Process Probe Table for Hash Join
(    2)    Insert Into Asynchronous Table Queue  ID = q1
           |  Broadcast to Coordinator Node
           |  Rows Can Overflow to Temporary Table

        Subsection #2:
(    9)    Access Table Name = DOOLE.PROJECT  ID = 2,7
           |  #Columns = 2
           |  Relation Scan
           |  |  Prefetch: Eligible
           |  Lock Intents
           |  |  Table: Intent Share
           |  |  Row  : Next Key Share
(    9)    |  Insert Into Asynchronous Table Queue  ID = q2
           |  |  Hash to Specific Node
           |  |  Rows Can Overflow to Temporary Tables
(    8)    Insert Into Asynchronous Table Queue Completion  ID = q2

        Subsection #3:
(    7)    Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
           |  #Columns = 3
           |  Relation Scan
           |  |  Prefetch: Eligible
           |  Lock Intents
           |  |  Table: Intent Share
           |  |  Row  : Next Key Share
(    7)    |  Insert Into Asynchronous Table Queue  ID = q3
           |  |  Hash to Specific Node
           |  |  Rows Can Overflow to Temporary Tables
(    6)    Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

                RETURN
                (   1)
                  |
                 BTQ
                (   2)
                  |
                HSJOIN
                (   3)
               /      \
          HSJOIN          DTQ
          (   4)        (   8)
         /      \         |
     TBSCAN      DTQ     TBSCAN
     (   5)    (   6)    (   9)
       |         |         |
     Table:    TBSCAN    Table:
     DOOLE     (   7)    DOOLE
```

```
 DEPARTMENT     |      PROJECT
            Table:
            DOOLE
            EMPLOYEE
```

This plan has all the same pieces as the plan in the first example, but the section has been broken into four subsections. The subsections have the following tasks:

- **Coordinator Subsection**. This subsection coordinates the other subsections. In this plan, it causes the other subsections to be distributed and then uses a table queue to gather the results to be returned to the application.
- **Subsection #1**. This subsection scans table queue q2 and uses a hash join to join it with the data from table queue q3. A second hash join then adds in the data from the DEPARTMENT table. The joined rows are then sent to the coordinator subsection using table queue q1.
- **Subsection #2**. This subsection scans the PROJECT table and hashes to a specific node with the results. These results are read by Subsection #1.
- **Subsection #3**. This subsection scans the EMPLOYEE table and hashes to a specific node with the results. These results are read by Subsection #1.

Example four: multipartition plan with inter-partition and intra-partition parallelism

This example shows the same SQL statement as the first example, but this query has been compiled on a partitioned database made up of three database partitions, each of which is on a four-way SMP machine.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/04
 Prep Time = 14:58:35

 Bind Timestamp = 2002-01-04-14.58.35.169555

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = Yes
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 145.198898
Estimated Cardinality = 474.720032

        Coordinator Subsection:
(-----)    Distribute Subsection #2
```

```
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(-----)     Distribute Subsection #3
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(-----)     Distribute Subsection #1
            |  Broadcast to Node List
            |  |  Nodes = 10, 33, 55
(    2)     Access Table Queue  ID = q1  #Columns = 5
(    1)     Return Data to Application
            |  #Columns = 5

         Subsection #1:
(    3)     Process Using 4 Subagents
(   10)     |  Access Table Queue  ID = q3  #Columns = 2
(    4)     |  Hash Join
            |  |  Estimated Build Size: 5737
            |  |  Estimated Probe Size: 8015
(    7)     |  |  Access Table Queue  ID = q5  #Columns = 3
(    5)     |  |  Hash Join
            |  |  |  Estimated Build Size: 5333
            |  |  |  Estimated Probe Size: 6421
(    6)     |  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
            |  |  |  |  #Columns = 3
            |  |  |  |  Parallel Scan
            |  |  |  |  Relation Scan
            |  |  |  |  |  Prefetch: Eligible
            |  |  |  |  Lock Intents
            |  |  |  |  |  Table: Intent Share
            |  |  |  |  |  Row  : Next Key Share
(    6)     |  |  |  |  Process Probe Table for Hash Join
(    3)     |  Insert Into Asynchronous Local Table Queue  ID = q2
(    3)     Access Local Table Queue  ID = q2  #Columns = 5
(    2)     Insert Into Asynchronous Table Queue  ID = q1
            |  Broadcast to Coordinator Node
            |  Rows Can Overflow to Temporary Table

         Subsection #2:
(   11)     Process Using 4 Subagents
(   12)     |  Access Table Name = DOOLE.PROJECT  ID = 2,7
            |  |  #Columns = 2
            |  |  Parallel Scan
            |  |  Relation Scan
            |  |  |  Prefetch: Eligible
            |  |  Lock Intents
            |  |  |  Table: Intent Share
            |  |  |  Row  : Next Key Share
(   11)     |  Insert Into Asynchronous Local Table Queue  ID = q4
(   11)     Access Local Table Queue  ID = q4  #Columns = 2
(   10)     Insert Into Asynchronous Table Queue  ID = q3
            |  Hash to Specific Node
            |  Rows Can Overflow to Temporary Tables

         Subsection #3:
(    8)     Process Using 4 Subagents
(    9)     |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
            |  |  #Columns = 3
            |  |  Parallel Scan
            |  |  Relation Scan
            |  |  |  Prefetch: Eligible
            |  |  Lock Intents
            |  |  |  Table: Intent Share
            |  |  |  Row  : Next Key Share
(    8)     |  Insert Into Asynchronous Local Table Queue  ID = q6
(    8)     Access Local Table Queue  ID = q6  #Columns = 3
(    7)     Insert Into Asynchronous Table Queue  ID = q5
            |  Hash to Specific Node
```

```
          |  Rows Can Overflow to Temporary Tables
End of section


Optimizer Plan:

              RETURN
              (   1)
                |
               BTQ
              (   2)
                |
               LTQ
              (   3)
                |
              HSJOIN
              (   4)
             /      \
       HSJOIN        DTQ
       (   5)       (  10)
      /      \        |
  TBSCAN     DTQ      LTQ
  (   6)    (   7)   (  11)
    |         |        |
Table:       LTQ     TBSCAN
DOOLE       (   8)   (  12)
DEPARTMENT    |        |
            TBSCAN   Table:
            (   9)   DOOLE
              |      PROJECT
           Table:
           DOOLE
           EMPLOYEE
```

This plan is similar to that in the third example, except that multiple subagents execute each subsection. Also, at the end of each subsection, a local table queue gathers the results from all of the subagents before the qualifying rows are inserted into the second table queue to be hashed to a specific node.

Example five: federated database plan

This example shows the same SQL statement as the first example, but this query has been compiled on a federated database where the tables DEPARTMENT and PROJECT are on a data source and the table EMPLOYEE is on the federated server.

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""

 Prep Date = 2002/01/11
 Prep Time = 13:52:48

 Bind Timestamp = 2002-01-11-13.52.48.325413

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"
```

```
-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:
  DECLARE EMPCUR CURSOR
  FOR
     SELECT e.lastname, e.job, d.deptname, d.location, p.projname
     FROM employee AS e, department AS d, project AS p
     WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 1804.625000
Estimated Cardinality = 112000.000000

(    7) Ship Distributed Subquery #2
        |  #Columns = 2
(    2) Hash Join
        |  Estimated Build Size: 48444
        |  Estimated Probe Size: 232571
(    6) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
        |  |  #Columns = 3
        |  |  Relation Scan
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    6) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
        |  |  Estimated Build Size: 7111
        |  |  Estimated Probe Size: 64606
(    4) |  |  Ship Distributed Subquery #1
        |  |  |  #Columns = 3
(    1) Return Data to Application
        |  #Columns = 5

Distributed Substatement #1:
(    4) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

             SELECT A0."DEPTNO", A0."DEPTNAME", A0."LOCATION"
             FROM "DOOLE"."DEPARTMENT" A0

        Nicknames Referenced:
           DOOLE.DEPARTMENT  ID = 32768
              Base = DOOLE.DEPARTMENT
        #Output Columns = 3

Distributed Substatement #2:
(    7) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

             SELECT A0."DEPTNO", A0."PROJNAME"
             FROM "DOOLE"."PROJECT" A0

        Nicknames Referenced:
           DOOLE.PROJECT  ID = 32769
              Base = DOOLE.PROJECT
        #Output Columns = 2

End of section

Optimizer Plan:

               RETURN
               (   1)
                 |
```

```
                HSJOIN
                (   2)
               /      \
         HSJOIN        SHIP
         (   3)       (   7)
        /      \         |
    SHIP      TBSCAN   Nickname:
   (   4)     (   6)   DOOLE
     |          |      PROJECT
 Nickname:   Table:
 DOOLE       DOOLE
 DEPARTMENT  EMPLOYEE
```

This plan has all the same pieces as the plan in the first example, except that the data for two of the tables are coming from data sources. The two tables are accessed through distributed subqueries which, in this case, simply select all the rows from those tables. Once the data is returned to the federated server, it is joined to the data from the local table.

Example six: XANDOR and XISCAN operators

This example shows how the XANDOR operator combines XISCAN scans of two individual indexes over XML data (*IDX1* and *IDX2*) that have been defined on the same table *XISCANTABLE*.

```
IBM DB2 Database SQL Explain Tool

****************** DYNAMIC ***************************************

================== STATEMENT ==========================================

        Isolation Level          = Cursor Stability
        Blocking                 = Block Unambiguous Cursors
        Query Optimization Class = 5

        Partition Parallel       = No
        Intra-Partition Parallel = No

        SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
  xquery
  for $c in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL ")/a[@x="1" ]/b[@y=
          "2" ] return $c

Section Code Page = 819

Estimated Cost = 192.266113
Estimated Cardinality = 1.800000

(    6) Index ANDing and ORing over XML
        |  Xpath is
        |  |  /child::element(a)[./child::element(b)
        |  |  /attribute::attribute(y)(:Index Search over XML 1:)
        |  |   and ./attribute::attribute(x)(:Index Search over XML 2:)
        |  |  ]
        |  Index Search over XML 1
        |  |  Access Table Name = ATTALURI.XISCANTABLE
        |  |  |  Index Scan over XML:  Name = ATTALURI.IDX1  ID = 6
        |  |  |  |  Physical Index over XML
        |  |  |  |  Index Columns:
        |  |  |  |  |  1: XMLCOL (Ascending)
        |  |  |  |  #Key Columns = 4
        |  |  |  |  Start Key: Inclusive Value
        |  |  |  |  |  |  1: ?
        |  |  |  |  |  |  2: ?
        |  |  |  |  |  |  3: ?
        |  |  |  |  |  |  4: ?
        |  |  |  |  Stop Key: Inclusive Value
        |  |  |  |  |  |  1: ?
```

```
        |  |  |  |  |  2: ?
        |  |  |  Index-Only Access
        |  |  |  Index Prefetch: None
        |  |  |  Isolation Level: Uncommitted Read
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent None
        |  |  |  |  Row  : None
        |  |  |  StopKey = StartKey
        |  |  |  Value Start Key = ?
        |  |  Index Search over XML 2
        |  |  |  Access Table Name = ATTALURI.XISCANTABLE
        |  |  |  Index Scan over XML:  Name = ATTALURI.IDX2  ID = 4
        |  |  |  |  Physical Index over XML
        |  |  |  |  Index Columns:
        |  |  |  |  |  1: XMLCOL (Ascending)
        |  |  |  #Key Columns = 4
        |  |  |  |  Start Key: Inclusive Value
        |  |  |  |  |  |  1: ?
        |  |  |  |  |  |  2: ?
        |  |  |  |  |  |  3: ?
        |  |  |  |  |  |  4: ?
        |  |  |  |  Stop Key: Inclusive Value
        |  |  |  |  |  |  1: ?
        |  |  |  |  |  |  2: ?
        |  |  |  Index-Only Access
        |  |  |  Index Prefetch: None
        |  |  |  Isolation Level: Uncommitted Read
        |  |  |  Lock Intents
        |  |  |  |  Table: Intent None
        |  |  |  |  Row  : None
        |  |  |  StopKey = StartKey
        |  |  |  Value Start Key = ?
(    5) Insert Into Sorted Temp Table  ID = t1
        |  #Columns = 1
        |  #Sort Key Columns = 1
        |  |  Key 1: (Ascending)
        |  Sortheap Allocation Parameters:
        |  |  #Rows     = 2
        |  |  Row Width = 16
        |  Piped
        |  Duplicate Elimination
(    4) List Prefetch Preparation
(    4) |  Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
        |  |  #Columns = 1
        |  |  Fetch Using Prefetched List
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    2) Nested Loop Join
        |  Piped Inner
(    9)  |  XML Doc Navigation
        |  |  Navigator is
        |  |  |  /fn:root($CONTEXT_NODE$())/child::element(a)(:#Xpath Predicates = 1:)
        |  |  |  [./child::element(b)(:Output nodeSeqRef :)
        |  |  |  (:#Xpath Predicates = 1:)
        |  |  |  /attribute::attribute(y) and
        |  |  |  ./attribute::attribute(x)]
(    1) Iterate over XML sequence for Xquery bindout
(    1) Return Data to Application
        |  #Columns = 1

End of section


Optimizer Plan:

                                    RETURN
                                    (   1)
                                      |
                                    NLJOIN
                                    (   2)
                                   /      \
                                FETCH    XSCAN
                               (----)    (   9)
                               /          \
                          RIDSCN           Table:
```

```
                          (   4)              ATTALURI
                            |                 XISCANTABLE
                           SORT
                          (   5)
                            |
                          XANDOR
                          (   6)
                     /---------/     \--------\
                  XISCAN                      XISCAN
                  (   6)                      (   6)
                 /       \                   /       \
            Index:        Table:          Index:       Table:
               ATTALURI      ATTALURI        ATTALURI     ATTALURI
               IDX1          XISCANTABLE     IDX2         XISCANTABLE
```

Example seven: XSCAN operator

This example shows how the XSCAN operator may appear in an access plan. This operator processes node references passed by a nested-loop join operator (NLJOIN). It is not represented with a direct input in the access plan.

```
IBM DB2 Database SQL Explain Tool

******************** DYNAMIC ***************************************

=================== STATEMENT ============================================

        Isolation Level          = Cursor Stability
        Blocking                 = Block Unambiguous Cursors
        Query Optimization Class = 5

        Partition Parallel       = No
        Intra-Partition Parallel = No

        SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
  xquery
  for $b in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL" )//book[position()<=
          2] return $b

Section Code Page = 819

Estimated Cost = 779592.625000
Estimated Cardinality = 540000000.000000

(    4) Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
         |  #Columns = 1
         |  Relation Scan
         |  |  Prefetch: Eligible
         |  Lock Intents
         |  |  Table: Intent Share
         |  |  Row  : Next Key Share
(    3) Nested Loop Join
         |  Piped Inner
(    6)  |  XML Doc Navigation
         |  |  Navigator is
         |  |  |  /fn:root($CONTEXT_NODE$())/descendant-or-self::node()(:Output nodeSeqRef :)
         |  |  |
(    5)  |  Nested Loop Join
         |  |  Piped Inner
(   11)  |  |  XML Doc Navigation
         |  |  |  Navigator is
         |  |  |  |  /fn:root($CONTEXT_NODE$())/child::element(book)(:Output nodeSeqRef :)
         |  |  |  |
(   10)  |  |  Aggregation
         |  |  |  Column Function(s)
(    9)  |  |  Nested Loop Join
         |  |  |  Piped Inner
(   12)  |  |  |  Unnest input XML sequence into stream of items with item number
(    8)  |  |  Nested Loop Join
         |  |  |  Piped Inner
(   14)  |  |  |  Table Constructor
         |  |  |  |  1-Row(s)
```

```
(    2) Nested Loop Join
        |  Piped Inner
(   16) |  Unnest input XML sequence into stream of items
(    1) Iterate over XML sequence for Xquery bindout
(    1) Return Data to Application
        |  #Columns = 1

End of section

Optimizer Plan:

                RETURN
                (   1)
                  |
                NLJOIN
                (   2)
              /-/      \-----\
          NLJOIN             *
          (   3)             *
         /       \           |
     TBSCAN       NLJOIN     TBSCAN
     (   4)       (   5)     (  16)
      |           /     \       |
    Table:      XSCAN   *     TFunc:
    ATTALURI    (   6)  *     SYSIBM
    XISCANTABLE         |     GENROW
                      NLJOIN
                      (   8)
                     /      \
                  NLJOIN     FILTER
                 (   9)      (  13)
                /      \       |
            GRPBY   TBSCAN  TBSCAN
            (  10)  (  12)  (  14)
              |       |       |
            XSCAN   TFunc:  TFunc:
            (  11)  SYSIBM  SYSIBM
                    GENROW  GENROW
```

Example eight: XISCAN operator

This example shows how the XISCAN operator scans the index over XML data *IDX1* defined on the table *XISCANTABLE*.

```
IBM DB2 Database SQL Explain Tool

******************** DYNAMIC ***************************************

=================== STATEMENT ============================================

        Isolation Level          = Cursor Stability
        Blocking                 = Block Unambiguous Cursors
        Query Optimization Class = 5

        Partition Parallel       = No
        Intra-Partition Parallel = No

        SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
  xquery
  for $c in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL ")/a[@x="1" ] return
          $c


Section Code Page = 819

Estimated Cost = 1666.833862
Estimated Cardinality = 18.000000
```

```
(    6) Access Table Name = ATTALURI.XISCANTABLE
        |  Index Scan over XML:  Name = ATTALURI.IDX1  ID = 4
        |  |  Physical Index over XML
        |  |  Index Columns:
        |  |  |  1: XMLCOL (Ascending)
        |  #Key Columns = 2
        |  |  Start Key: Inclusive Value
        |  |  |  |  1: ?
        |  |  |  |  2: ?
        |  |  Stop Key: Inclusive Value
        |  |  |  |  1: ?
        |  |  |  |  2: ?
        |  Index-Only Access
        |  Index Prefetch: None
        |  Isolation Level: Uncommitted Read
        |  Lock Intents
        |  |  Table: Intent None
        |  |  Row  : None
        |  StopKey = StartKey
        |  Value Start Key = ?
        |  Xpath is
        |  |  /child::element(a)/attribute::attribute(x)
(    5) Insert Into Sorted Temp Table  ID = t1
        |  #Columns = 1
        |  #Sort Key Columns = 1
        |  |  Key 1: (Ascending)
        |  Sortheap Allocation Parameters:
        |  |  #Rows     = 18
        |  |  Row Width = 16
        |  Piped
        |  Duplicate Elimination
(    4) List Prefetch Preparation
(    4) |  Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
        |  |  #Columns = 1
        |  |  Fetch Using Prefetched List
        |  |  |  Prefetch: Eligible
        |  |  Lock Intents
        |  |  |  Table: Intent Share
        |  |  |  Row  : Next Key Share
(    2) Nested Loop Join
        |  Piped Inner
(    7) |  XML Doc Navigation
        |  |  Navigator is
        |  |  |  /fn:root($CONTEXT_NODE$())/child::element(a)(:Output nodeSeqRef :)
        |  |  |  (:#Xpath Predicates = 1:)
        |  |  |  /attribute::attribute(x)
(    1) Iterate over XML sequence for Xquery bindout
(    1) Return Data to Application
        |  #Columns = 1

End of section


Optimizer Plan:

                    RETURN
                    (   1)
                      |
                    NLJOIN
                    (   2)
                   /      \
                FETCH     XSCAN
                (----)    (   7)
               /      \
           RIDSCN     Table:
           (   4)     ATTALURI
             |        XISCANTABLE
```

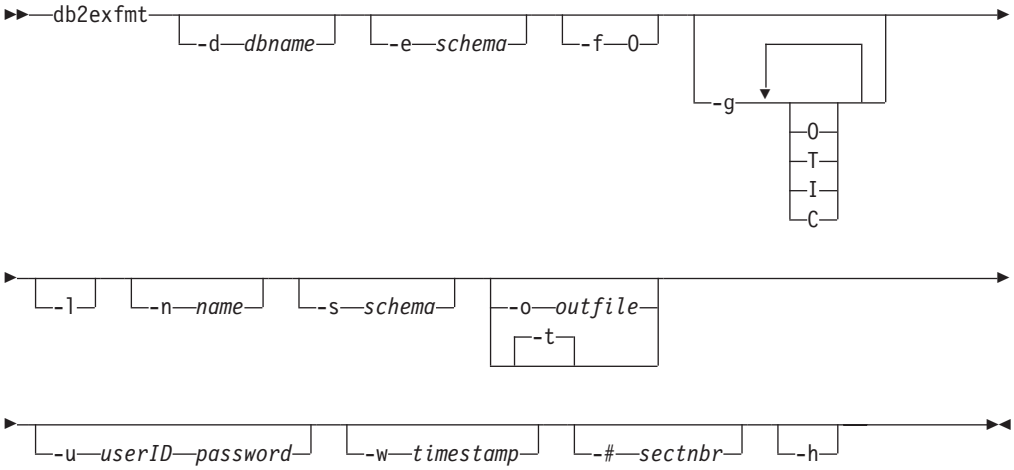
```
       SORT
       (   5)
         |
       XISCAN
       (   6)
      /      \
Index:       Table:
ATTALURI     ATTALURI
IDX1         XISCANTABLE
```

Appendix F. db2exfmt - Explain table format

You use the db2exfmt tool to format the contents of the explain tables. This tool is located in the `misc` subdirectory of the instance `sqllib` directory.

To use the tool, you require read access to the explain tables being formatted.

Command syntax:



Command parameters:

-d *dbname*
Name of the database containing packages.

-e *schema*
Explain table SQL schema.

-f Formatting flags. In this release, the only supported value is `O` (operator summary).

-g Graph plan. If only `-g` is specified, a graph, followed by formatted information for all of the tables, is generated. Otherwise, any combination of the following valid values can be specified:
- **O** Generate a graph only. Do not format the table contents.
- **T** Include total cost under each operator in the graph.
- **I** Include I/O cost under each operator in the graph.
- **C** Include the expected output cardinality (number of tuples) of each operator in the graph.

-l Respect case when processing package names.

-n *name*
Name of the source of the explain request (SOURCE_NAME).

-s *schema*
SQL schema or qualifier of the source of the explain request (SOURCE_SCHEMA).

-o *outfile*
Output file name.

-t Direct the output to the terminal.

-u *userID password*
When connecting to a database, use the provided user ID and password.

Both the user ID and password must be valid according to naming conventions and be recognized by the database.

-w *timestamp*
Explain time stamp. Specify `-1` to obtain the latest explain request.

-# *sectnbr*
Section number in the source. To request all sections, specify zero.

-h Display help information. When this option is specified, all other options are ignored, and only the help information is displayed.

Usage notes:

You will be prompted for any parameter values that are not supplied, or that are incompletely specified, except in the case of the `-h` and the `-l` options.

If an explain table SQL schema is not provided, the value of the environment variable **USER** is used as the default. If this variable is not found, the user is prompted for an explain table SQL schema.

Source name, source SQL schema, and explain time stamp can be supplied in LIKE predicate form, which allows the percent sign (%) and the underscore (_) to be used as pattern matching characters to select multiple sources with one invocation. For the latest explained statement, the explain time can be specified as `-1`.

If `-o` is specified without a file name, and `-t` is not specified, the user is prompted for a file name (the default name is `db2exfmt.out`). If neither `-o` nor `-t` is specified, the user is prompted for a file name (the default option is terminal output). If `-o` and `-t` are both specified, the output is directed to the terminal.

Related concepts:

- "Explain tools" on page 222
- "Guidelines for capturing explain information" on page 231
- "Guidelines for using explain information" on page 223

Appendix G. Using the Windows Performance Monitor

Windows performance monitor introduction

When working with DB2 database manager for Windows, there are tools that can be used to monitor performance:

- **DB2 Performance Expert**

  DB2 Performance Expert for Multiplatforms, Version 1.1 consolidates, reports, analyzes and recommends self-managing and resource tuning changes based on DB2 database performance-related information.
- **DB2 Health Center**

  The functions of the Health Center provide you with different methods to work with performance-related information. These functions somewhat replace the performance monitor capability of the Control Center.
- **Windows Performance Monitor**

  The Windows Performance Monitor enables you to monitor both database and system performance, retrieving information from any of the performance data providers registered with the system. Windows also provides performance information data on all aspects of computer operation including:
  - CPU usage
  - Memory utilization
  - Disk activity
  - Network activity

Related tasks:

- "Accessing remote DB2 database performance information" on page 674
- "Displaying DB2 database and DB2 Connect performance values" on page 673
- "Enabling remote access to DB2 performance information" on page 672
- "Registering DB2 with the Windows performance monitor" on page 671
- "Resetting DB2 performance values" on page 674

Related reference:

- "Windows performance objects" on page 673

Registering DB2 with the Windows performance monitor

Procedure:

The setup program automatically registers DB2 with the Windows Performance Monitor for you.

To make DB2 database and DB2 Connect performance information accessible to the Windows Performance Monitor, you must register the DLL for the DB2 for Windows Performance Counters. This also enables any other Windows application using the Win32 performance APIs to get performance data.

To install and register the DB2 for Windows Performance Counters DLL (DB2Perf.DLL) with the Windows Performance Monitor, type:

```
db2perfi -i
```

Registering the DLL also creates a new key in the services option of the registry. One entry gives the name of the DLL, which provides the counter support. Three other entries give names of functions provided within that DLL. These functions include:

- **Open**

  Called when the DLL is first loaded by the system in a process.
- **Collect**

  Called to request performance information from the DLL.
- **Close**

  Called when the DLL is unloaded.

Related reference:

- "db2perfi - Performance counters registration utility command" in *Command Reference*

Enabling remote access to DB2 performance information

Procedure:

If your DB2 for Windows workstation is networked to other Windows computers, you can use the feature described in this section.

In order to see Windows performance objects from another DB2 for Windows computer, you must register an administrator username and password with the DB2 database manager. (The default Windows Performance Monitor username, **SYSTEM**, is a DB2 database reserved word and cannot be used.) To register the name, type:

```
db2perfr -r username password
```

Note: The `username` used must conform to the DB2 database naming rules.

The username and password data is held in a key in the registry, with security that allows access only by administrators and the SYSTEM account. The data is encoded to prevent security concerns about storing an administrator password in the registry.

Notes:

1. Once a username and password combination has been registered with the DB2 database system, even local instances of the Performance Monitor will explicitly log on using that username and password. This means that if the username information registered with DB2 database system does not match, local sessions of the Performance Monitor will not show DB2 database performance information.
2. The username and password combination must be maintained to match the username and password values stored in the Windows Security database. If the username or password is changed in the Windows Security database, the username and password combination used for remote performance monitoring must be reset.
3. To deregister, type:

   ```
   db2perfr -u <username> <password>
   ```

Related concepts:

- "General naming rules" in *Administration Guide: Implementation*

Related reference:

- "db2perfr - Performance monitor registration tool command" in *Command Reference*

Displaying DB2 database and DB2 Connect performance values

Procedure:

To display DB2 database and DB2 Connect performance values using the Performance Monitor, simply choose the performance counters whose values you want displayed from the **Add to** box. This box displays a list of performance objects providing performance data. Select an object to see a list of the counters it supplies.

A performance object can also have multiple instances. For example, the LogicalDisk object provides counters such as "% Disk Read Time" and "Disk Bytes/sec"; it also has an instance for each logical drive in the computer, including "C:" and "D:".

Related concepts:

- "Windows performance monitor introduction" on page 671

Related reference:

- "Windows performance objects" on page 673

Windows performance objects

Windows provides the following performance objects:

- **DB2 Database Manager**

  This object provides general information for a single Windows instance. The DB2 database instance being monitored appears as the object instance.

  For practical and performance reasons, you can only get performance information from one DB2 database instance at a time. The DB2 database instance that the Performance Monitor shows is governed by the `db2instance` registry variable in the Performance Monitor process. If you have multiple DB2 database instances running simultaneously and want to see performance information from more than one, you must start a separate session of the Performance Monitor, with `db2instance` set to the relevant value for each DB2 database instance to be monitored.

  If you are running a partitioned database environment, you can only get performance information from one database partition server at a time. By default, the performance information for the default database partition (that is, the database partition that has logical port 0) is displayed. To see performance information of another database partition, you must start a separate session of the Performance Monitor with the DB2NODE environment variable set to the database partition number of the database partition to be monitored.

- **DB2 Databases**

  This object provides information for a particular database. Information is available for each currently active database.

- **DB2 Applications**

  This object provides information for a particular DB2 database application. Information is available for each currently active DB2 database application.
- **DB2 DCS Databases**

  This object provides information for a particular DCS database. Information is available for each currently active database.
- **DB2 DCS Applications**

  This object provides information for a particular DB2 DCS application. Information is available for each currently active DB2 DCS application.

Which of these objects will be listed by the Windows Performance Monitor depends on what is installed on your Windows computer and what applications are active. For example, if the DB2 database manager is installed has been started, the DB2 Database Manager object will be listed. If there are also some DB2 databases and applications currently active on that computer, the DB2 Databases and DB2 Applications objects will be listed as well. If you are using your Windows system as a DB2 Connect gateway and there are some DCS databases and applications currently active, the DB2 DCS Databases and DB2 DCS Applications objects will be listed.

Accessing remote DB2 database performance information

Procedure:

Enabling remote access to DB2 Performance Information was discussed earlier. In the **Add to** box, select another computer to monitor. This brings up a list of all the available performance objects on that computer.

In order to be able to monitor DB2 Performance object on a remote computer, the level of the DB2 database or DB2 Connect code installed on that computer must be Version 6 or higher.

Related concepts:

• "Windows performance monitor introduction" on page 671

Resetting DB2 performance values

Procedure:

When an application calls the DB2 monitor APIs, the information returned is normally the cumulative values since the DB2 database server was started. However, often it is useful to:

- Reset performance values
- Run a test
- Reset the values again
- Re-run the test.

To reset database performance values, use the **db2perfc** program. Type:

```
db2perfc
```

By default, this resets performance values for all active DB2 databases. However, you can also specify a list of databases to reset. You can also use the -d option to specify that performance values for DCS databases should be reset. For example:

```
db2perfc
db2perfc dbalias1 dbalias2 ... dbaliasn

db2perfc  -d
db2perfc  -d  dbalias1 dbalias2 ... dbaliasn
```

The first example resets performance values for all active DB2 databases. The next example resets values for specific DB2 databases. The third example resets performance values for all active DB2 DCS databases. The last example resets values for specific DB2 DCS databases.

The **db2perfc** program resets the values for ALL programs currently accessing database performance information for the relevant DB2 database server instance (that is, the one held in DB2INSTANCE in the session in which you run **db2perfc**.

Invoking **db2perfc** also resets the values seen by anyone remotely accessing DB2 database performance information when the **db2perfc** command is executed.

Note: There is a DB2 database API, sqlmrset, that allows an application to reset the values it sees locally, not globally, for particular databases.

Related reference:

- "db2perfc - Reset database performance values command" in *Command Reference*
- "db2ResetMonitor API - Reset the database system monitor data" in *Administrative API Reference*

Appendix H. DB2 Database technical information

Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF CD)
  - printed books
- Command line help
  - Command help
  - Message help
- Sample programs

IBM periodically makes documentation updates available. If you access the online version on the DB2 Information Center at ibm.com®, you do not need to install documentation updates because this version is kept up-to-date by IBM. If you have installed the DB2 Information Center, it is recommended that you install the documentation updates. Documentation updates allow you to update the information that you installed from the *DB2 Information Center CD* or downloaded from Passport Advantage as new information becomes available.

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and Redbooks™ online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how we can improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

Related concepts:

- "Features of the DB2 Information Center" in *Online DB2 Information Center*
- "Sample files" in *Samples Topics*

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*
- "Updating the DB2 Information Center installed on your computer or intranet server" on page 683

Related reference:

- "DB2 technical library in PDF format" on page 678

DB2 technical library in PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order.

Although the tables identify books available in print, the books might not be available in your country or region.

The information in these books is fundamental to all DB2 users; you will find this information useful whether you are a programmer, a database administrator, or someone who works with DB2 Connect or other DB2 products.

Table 98. DB2 technical information

Name	Form Number	Available in print
Administration Guide: Implementation	SC10-4221	Yes
Administration Guide: Planning	SC10-4223	Yes
Administrative API Reference	SC10-4231	Yes
Administrative SQL Routines and Views	SC10-4293	No
Call Level Interface Guide and Reference, Volume 1	SC10-4224	Yes
Call Level Interface Guide and Reference, Volume 2	SC10-4225	Yes
Command Reference	SC10-4226	No
Data Movement Utilities Guide and Reference	SC10-4227	Yes
Data Recovery and High Availability Guide and Reference	SC10-4228	Yes
Developing ADO.NET and OLE DB Applications	SC10-4230	Yes
Developing Embedded SQL Applications	SC10-4232	Yes
Developing SQL and External Routines	SC10-4373	No

Table 98. DB2 technical information (continued)

Name	Form Number	Available in print
Developing Java Applications	SC10-4233	Yes
Developing Perl and PHP Applications	SC10-4234	No
Getting Started with Database Application Development	SC10-4252	Yes
Getting started with DB2 installation and administration on Linux and Windows	GC10-4247	Yes
Message Reference Volume 1	SC10-4238	No
Message Reference Volume 2	SC10-4239	No
Migration Guide	GC10-4237	Yes
Net Search Extender Administration and User's Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6842	Yes
Performance Guide	SC10-4222	Yes
Query Patroller Administration and User's Guide	GC10-4241	Yes
Quick Beginnings for DB2 Clients	GC10-4242	No
Quick Beginnings for DB2 Servers	GC10-4246	Yes
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC18-9749	Yes
SQL Guide	SC10-4248	Yes
SQL Reference, Volume 1	SC10-4249	Yes
SQL Reference, Volume 2	SC10-4250	Yes
System Monitor Guide and Reference	SC10-4251	Yes
Troubleshooting Guide	GC10-4240	No
Visual Explain Tutorial	SC10-4319	No
What's New	SC10-4253	Yes
XML Extender Administration and Programming	SC18-9750	Yes
XML Guide	SC10-4254	Yes
XQuery Reference	SC18-9796	Yes

Table 99. DB2 Connect-specific technical information

Name	Form Number	Available in print
DB2 Connect User's Guide	SC10-4229	Yes
Quick Beginnings for DB2 Connect Personal Edition	GC10-4244	Yes

Table 99. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Servers	GC10-4243	Yes

Table 100. WebSphere Information Integration technical information

Name	Form Number	Available in print
WebSphere Information Integration: Administration Guide for Federated Systems	SC19-1001	Yes
WebSphere Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1000	Yes
WebSphere Information Integration: Configuration Guide for Federated Data Sources	SC19-1034	No
WebSphere Information Integration: SQL Replication Guide and Reference	SC19-1002	Yes

Note: The DB2 Release Notes provide additional information specific to your product's release and fix pack level. For more information, see the related links.

Related concepts:

- "Overview of the DB2 technical information" on page 677
- "About the Release Notes" in *Release notes*

Related tasks:

- "Ordering printed DB2 books" on page 680

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* CD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation CD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation CD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Procedure:

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  - Locate the contact information for your local representative from one of the following Web sites:
    - The IBM directory of world wide contacts at www.ibm.com/planetwide
    - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the "About this site" link.
  - When you call, specify that you want to order a DB2 publication.
  - Provide your representative with the titles and form numbers of the books that you want to order.

Related concepts:

- "Overview of the DB2 technical information" on page 677

Related reference:

- "DB2 technical library in PDF format" on page 678

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure:

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*

Accessing different versions of the DB2 Information Center

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Related tasks:

- "Updating the DB2 Information Center installed on your computer or intranet server" on page 683

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

Procedure:

To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools** —> **Internet Options** —> **Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button.

     Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

To display topics in your preferred language in a Firefox or Mozilla browser:

1. Select the **Tools** —> **Options** —> **Languages** button. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Related concepts:

- "Overview of the DB2 technical information" on page 677

Updating the DB2 Information Center installed on your computer or intranet server

If you have a locally-installed DB2 Information Center, updated topics can be available for download. The 'Last updated' value found at the bottom of most topics indicates the current level for that topic.

To determine if there is an update available for the entire DB2 Information Center, look for the 'Last updated' value on the Information Center home page. Compare the value in your locally installed home page to the latest value which is available on the IBM hosted Information Center home page. If they are the same, you have the latest documentation level and no update is required. If the are not the same, you should update your locally-installed Information Center.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to determine if update packages are available from IBM. If update packages are available, use the Update feature to download the packages. (The Update feature is only available in stand-alone mode.)
3. Stop the stand-alone Information Center, and restart the DB2 Information Center service on your computer.

Procedure:

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center service.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv9 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `C:\Program Files\IBM\DB2 Information Center\Version 9` directory.
     c. Run the `help_start.bat` file using the fully qualified path for the DB2 Information Center:
        ```
        <DB2 Information Center dir>\doc\bin\help_start.bat
        ```
   - On Linux:
     a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9` directory.
     b. Run the `help_start.sh` file using the fully qualified path for the DB2 Information Center:
        ```
        <DB2 Information Center dir>/doc/bin/help_start
        ```

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the Update button (). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click **Install Updates**.
5. After the download and installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center.
   - On Windows, run the `help_end.bat` file using the fully qualified path for the DB2 Information Center:
     ```
     <DB2 Information Center dir>\doc\bin\help_end.bat
     ```
   - On Linux, run the `help_end.sh` file using the fully qualified path for the DB2 Information Center:
     ```
     <DB2 Information Center dir>/doc/bin/help_end
     ```
7. Restart the DB2 Information Center service.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:
     ```
     /etc/init.d/db2icdv9 start
     ```

The updated DB2 Information Center displays the new and updated topics.

Related concepts:

- "DB2 Information Center installation options" in *Quick Beginnings for DB2 Servers*

Related tasks:

- "Installing the DB2 Information Center using the DB2 Setup wizard (Linux)" in *Quick Beginnings for DB2 Servers*
- "Installing the DB2 Information Center using the DB2 Setup wizard (Windows)" in *Quick Beginnings for DB2 Servers*

DB2 Visual Explain tutorial

The DB2 Visual Explain tutorial helps you learn about analyzing, optimizing, and tuning SQL statements for better performance. Lessons provide step-by-step instructions.

Before you begin:

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 Visual Explain tutorial:

To view the tutorial, click on the title.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

Related concepts:

- "Visual Explain overview" in *Administration Guide: Implementation*

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation

Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site

Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Related concepts:

- "Introduction to problem determination" in *Troubleshooting Guide*
- "Overview of the DB2 technical information" on page 677

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

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.

Appendix I. Notices

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

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

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

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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

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

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 that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

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

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

Trademarks

Company, product, or service names identified in the documents of the DB2 Version 9 documentation library may be trademarks or service marks of International Business Machines Corporation or other companies. Information on the trademarks of IBM Corporation in the United States, other countries, or both is located at http://www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Itanium®, Pentium®, and Xeon® are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Contacting IBM

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts at http://www.ibm.com/planetwide

To learn more about DB2 products, go to http://www.ibm.com/software/data/db2/.

Printed in USA

SC10-4222-00

Spine information:

IBM DB2 9 BETA

DB2 Version 9

Performance Guide