RUNSTATS in DB2 UDB Version 8.2
Guidelines and examples

Larry Pay
December 02, 2004

RUNSTATS is the utility in DB2® Universal Database™ (DB2 UDB) that updates catalog statistics used by the optimizer to determine the fastest path to your data. With DB2 UDB V8.2, the number of RUNSTATS options have increased considerably. Understanding how to use RUNSTATS, with both the existing options and the new ones, can help you gain optimal performance for your database. The author includes many examples to help make it all clear.

Why is RUNSTATS important?

Almost every major database today uses some method of updating catalog statistics to provide the best information possible to its optimizer. Think of the optimizer as an explorer waiting to navigate a mountain represented by the data in the system. Updating the catalog statistics would provide the optimizer with the most current map to navigate quickly through the terrain.

The DB2 optimizer uses catalog statistics to determine the best access path, and the main method employed to update these catalog statistics is to run the RUNSTATS utility. When there is any data modification to the user tables, the catalog statistics tables are not automatically modified. A RUNSTATS command has to be performed on tables and indexes so that columns in the catalog tables are updated with the latest information.

The following catalog columns can be queried to determine if RUNSTATS has been performed on the tables and indexes:

- If the CARD column of the SYSCAT.TABLES view displays a value of -1, or the STATS_TIME column displays a NULL value for a table, then the RUNSTATS utility has not run for that table.
- If the NLEAVE, NLEVELS and FULLKEYCARD columns of the SYSCAT.INDEXES view display a value of -1, or the STATS_TIME column displays a NULL value for an index, then the RUNSTATS utility has not run for that index.

After data is initially loaded to a table, subsequent changes in the form of updates, deletes, and inserts to tables may be inevitable. Indexes may be added or dropped. Data distribution in particular columns may change over time. None of these changes are reflected in the catalog,
unless RUNSTATS is performed on these tables and indexes. Catalog statistics can become stale over time as changes to the table and the data occur.

Performing RUNSTATS is important, but running it often enough may be an issue. For small tables, issuing the RUNSTATS command would be a trivial task. However, as the table grows, the RUNSTATS command takes more time, CPU and memory resources to complete. Eventually, you must consider the tradeoff of allocating more time and resources to RUNSTATS versus the potential degradation in performance if you do not run it. It is usually only when critical queries start slowing down that administrators give RUNSTATS the proper attention. You can avoid a knee-jerk reaction to tuning queries and performing RUNSTATS by formulating a strategy for collecting statistics efficiently and effectively.

**Collect catalog statistics using RUNSTATS**

You should ideally perform RUNSTATS on tables and indexes in the following situations:

- After data has been loaded into a table and the appropriate indexes have been created. However, it is preferable to create indexes before performing the LOAD command, and collecting statistics during the LOAD.
- After a new index has been created.
- After a table has been reorganized with the REORG utility.
- After the table and its indexes have been extensively updated by data updates, deletions, and insertions.
- After the prefetch size has been changed.
- After running the REDISTRIBUTE DATABASE PARTITION GROUP utility.

You can determine the effects of running RUNSTATS on the access plans by comparing the EXPLAIN outputs of the queries before and after RUNSTATS.

After the completion of every RUNSTATS statement, you should execute an explicit COMMIT WORK. The COMMIT releases locks and avoids filling up the log when gathering statistics on multiple tables.

---

Note: In RUNSTATS syntax, you must use the fully qualified table name, `schema.table-name`, and the fully qualified index name, `schema.index-name`. You can perform RUNSTATS on all columns or only on some columns, or column groups (except LONG and LOB columns). The clause ON ALL COLUMNS is the default if no column specific clauses are specified.

Rebind packages containing static SQL using the BIND command, or the REBIND command (and optionally re-explain their statements) after collecting statistics with RUNSTATS. The `db2rbind` command can be used to rebind all packages in the database. Use the FLUSH PACKAGE command to remove all cached dynamic SQL statements currently in the package cache and force the next request to be implicitly compiled.

The following examples illustrate using RUNSTATS to collect statistics:

**Example 1: Collect statistics on all columns**

```sql
RUNSTATS ON TABLE db2admin.department ON ALL COLUMNS
```
This is equivalent to:

**RUNSTATS ON TABLE db2admin.department**

**Example 2: Collect catalog statistics on individual column names**

**RUNSTATS ON TABLE db2admin.department ON COLUMNS (deptno, deptname)**

Example 3 shows how you collect statistics on key columns. The phrase "key columns" means the columns that make up the indexes defined on the table. If there are no indexes, no column statistics are collected.

**Example 3: Collect catalog statistics on key columns**

**RUNSTATS ON TABLE db2admin.department ON KEY COLUMNS**

**Example 4: Collect catalog statistics on key columns and on one column that is not**

**RUNSTATS ON TABLE db2admin.department ON KEY COLUMNS AND COLUMNS (deptname)**

**Example 5: Collect catalog statistics on the table and indexes without distribution statistics**

**RUNSTATS ON TABLE db2admin.department AND INDEXES ALL**

**Example 6: Collect catalog statistics on the table and enhanced statistics on the indexes, but without distribution statistics**

**RUNSTATS ON TABLE db2admin.department AND DETAILED INDEXES ALL**

**Example 7: Collect catalog statistics on 3 specific indexes only (no table statistics)**

**RUNSTATS ON TABLE db2admin.department FOR INDEXES db2admin.INX1, db2admin.INX2, db2admin.INX3**

**Example 8: Collect catalog statistics on all indexes only**

**RUNSTATS ON TABLE db2admin.department FOR INDEXES ALL**

If new indexes have been created, and the corresponding table has not been modified since the last RUNSTATS execution, then you can collect catalog statistics on indexes only. It is also possible to collect catalog statistics while the index is being created. I'll discuss this method in the section **Alternative ways of collecting catalog statistics.**

**Use RUNSTATS WITH DISTRIBUTION**

When you have determined that the table contains data that is not uniformly distributed, run RUNSTATS with the WITH DISTRIBUTION clause. Normally, the catalog statistics tables contain
information about the highest and lowest values in a table, and the optimizer assumes that the data values are evenly distributed between these end-point values. However, if data values are very different from one another, or data values are clustered together in some spots, or many duplicate data values are encountered - the optimizer cannot choose an optimal access path unless distribution statistics are collected. Using the WITH DISTRIBUTION clause can also help queries with predicates that do not have parameter markers or host variables, because the optimizer still doesn't know if the run-time value will have many or few rows.

The following examples illustrate the different ways of using RUNSTATS to collect statistics with distribution:

**Example 9: Collect catalog statistics on the table and indexes with distribution statistics**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
```

**Example 10: Collect catalog statistics on the table and enhanced statistics on the indexes, as well as distribution statistics**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND DETAILED INDEXES ALL
```

**Example 11: Collect catalog statistics on selected columns with distribution**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS (deptno, deptname)
```

**Example 12: Collect catalog statistics on only the table, with basic column statistics for deptno and deptname and distribution statistics on mgrno and admrdept**

```
RUNSTATS ON TABLE db2admin.department ON COLUMNS (deptno, deptname) WITH DISTRIBUTION ON COLUMNS (mgrno, admrdept)
```

**Example 13: Collect catalog statistics with distribution on all columns that are part of the index, plus two columns that are not**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON KEY COLUMNS AND COLUMNS (admrdept, location)
```

**RUNSTATS with frequency and quantile statistics**

When you perform RUNSTATS with the WITH DISTRIBUTION clause, a set of frequency and quantile statistics is chosen, depending on the options given in the RUNSTATS command.

RUNSTATS collects two types of data distribution statistics: frequency statistics and quantile statistics.

The frequency statistics default value is controlled by the `num_freqvalues` database configuration parameter, and provides information about the column and the data values with the highest
number of duplicates. The default value is 10, and the recommended setting should be between 10 and 100. If `num_freqvalues` is set to zero, no frequent value statistics are retained.

The quantile statistics default value is controlled by the `num_quantiles` database configuration parameter and provides information about how data values are distributed in relation to other values. The `num_quantiles` database configuration parameter specifies the number of sections into which the column data values should be grouped. The default value is 20, and the recommended range for this parameter is between 20 and 50. If you set this parameter to zero or "1", no quantile statistics are collected.

If `num_freqvalues` and `num_quantiles` are not specified on either the RUNSTATS command column or table level, the `num_freqvalues` value is taken from the `num_freqvalues` database configuration parameter, while the `num_quantiles` value is taken from the `num_quantiles` database configuration parameter.

The precision for the frequency and quantile statistics can be modified for an individual column or a group of columns. Increasing the precision of distribution statistics leads to greater CPU and memory consumption, as well as increased catalog space. Consider only the most important columns for the most important queries that have selection predicates for these distribution statistics.

RUNSTATS will not collect distribution statistics when any of the following conditions are present:

- When the `num_freqvalues` configuration parameter is set to zero (0) and the `num_quantiles` database configuration parameter is set to zero (0) or 1
- When each data value is unique
- When the column is LONG, LOB or a structured column
- If there is only one non-NULL value in the column
- Extended indexes or declared temporary tables

The following examples illustrate the different ways of using RUNSTATS to collect catalog statistics and specifying `num_freqvalues` and `num_quantiles`:

Assume that the `num_freqvalues` database configuration parameter value is 10, and the `num_quantiles` database configuration parameter value is 20.

**Example 14. Collect catalog statistics with distribution**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL
```

The `num_freqvalues` parameter is set to 10, and the `num_quantiles` parameter is set to 20, since no `num_freqvalues` or `num_quantiles` parameters were specified on the command line.

**Example 15: Collect catalog statistics on the table only, on all columns with distribution statistics using a specified num_freqvalues while picking the num_quantiles from the database configuration setting**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION DEFAULT
  NUM_FREQVALUES 40
```
The `num_freqvalues` parameter is set to 40, and the `num_quantiles` parameter is set to 20.

**Example 16:** Collect catalog statistics on table, with distribution statistics on columns deptno and deptname. Distribution statistics limits are set individually for the deptname column, while deptno column uses a common default. Catalog statistics are also collected for two indexes IDX1 and IDX2.

```sql
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
  (deptno, deptname NUM_FREQVALUES 50 NUM_QUANTILES 100) DEFAULT
  NUM_FREQVALUES 5 NUM_QUANTILES 10 AND INDEXES db2admin.IDX1, db2admin.IDX2
```

For the deptname column, `num_freqvalues` is 50 and `num_quantiles` is 100.
For the deptno column, `num_freqvalues` is 5 and `num_quantiles` is 10.

**Example 17:** Collect catalog statistics on all indexes and on column deptname with distribution. Distribution statistics for columns not listed will be cleared.

```sql
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
  (deptname NUM_FREQVALUES 20 NUM_QUANTILES 40)
  AND INDEXES ALL
```

For the deptname column, `num_freqvalues` is 20 and `num_quantiles` is 40.

**Example 18:** Collect catalog statistics on all indexes and on columns deptno and deptname. The `num_freqvalues` and `num_quantiles` values for the deptno column will be obtained from the default.

```sql
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON COLUMNS
  (deptno, deptname NUM_FREQVALUES 20 NUM_QUANTILES 40) DEFAULT
  NUM_FREQVALUES 0 NUM_QUANTILES 0 AND INDEXES ALL
```

For the deptname column, `num_freqvalues` is 20, and `num_quantiles` is 40.
For the deptno column, `num_freqvalues` is 0, and `num_quantiles` is 0.
All other columns will not have any distribution statistics.

**RUNSTATS with column group statistics**

Column Group Statistics obtains the number of distinct combinations of values for a group of columns. In general, the basic statistics available to the DB2 optimizer does not detect data correlation. The use of column groups provides more accurate estimates of the joint selectivity of multiple predicates. Column group statistics assume data is uniform; distribution statistics are not available yet on column groups.

The product of individual column cardinalities is compared to the cardinality of the column group to get better correlation estimates.

The following example illustrates using RUNSTATS to collect catalog statistics that capture column group information:
Example 19: Collect catalog statistics that capture column groups

RUNSTATS ON TABLE db2admin.department ON COLUMNS ((deptno, deptname),
  deptname, mrgno, (admrdept, location))

In this case, there are two column groups: (deptno, deptname) and (admrdept, location).

RUNSTATS with LIKE STATISTICS

When the LIKE STATISTICS clause is specified in RUNSTATS, additional column statistics are collected. These statistics are stored in the SUB_COUNT and SUB_DELIM_LENGTH columns in the SYSIBM.SYSCOLUMNS table. They are collected for string columns only, and they are used by the query optimizer to improve the selectivity estimates for predicates of the type "column LIKE '%abc'" and "column LIKE '%abc%'".

The following example illustrates using RUNSTATS to collect catalog statistics that capture LIKE statistics:

Example 20: Collect catalog statistics on all columns and LIKE statistics on a specific VARCHAR column

RUNSTATS ON TABLE db2admin.department ON ALL COLUMNS and COLUMNS (deptname LIKE STATISTICS)

RUNSTATS with a statistics profile

In DB2 V8.2, it is now possible to set up a statistics profile for RUNSTATS. A statistics profile is a set of options that predefine which statistics are to be collected on a particular table.

A statistics profile is registered or stored in the table descriptor and system catalogs when the command parameter "SET PROFILE" is added to the RUNSTATS command. To update this statistics profile, use the command parameter "UPDATE PROFILE".

There is no delete profile option.

The following examples show how this feature is used:

Example 21: Register a statistics profile only without collecting catalog statistics

RUNSTATS ON TABLE db2admin.department AND INDEXES ALL SET PROFILE ONLY

The clause "SET PROFILE ONLY" for RUNSTATS does not collect statistics.

Example 22: Register a statistics profile and execute the RUNSTATS command options of the stored statistics profile to collect catalog statistics

RUNSTATS in DB2 UDB Version 8.2
Example 23: Modify an existing statistics profile only; no catalog statistics are collected

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL UPDATE PROFILE ONLY
```

Example 24: Modify an existing statistics profile and execute the RUNSTATS command options of the updated statistics profile to collect catalog statistics

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION AND INDEXES ALL UPDATE PROFILE
```

Example 25: Query the RUNSTATS options corresponding to the previously registered statistics profile

```
SELECT STATISTICS_PROFILE FROM SYSIBM.SYSTABLES WHERE NAME = 'DEPARTMENT' AND CREATOR = 'DB2ADMIN'
```

Example 26: Collect catalog statistics using a previously registered statistics profile

```
RUNSTATS ON TABLE db2admin.department USE PROFILE
```

**RUNSTATS with sampling**

With databases growing at an unrelenting pace, the ability to collect statistics by accessing all of the data may be hampered by fixed batch windows or memory and CPU constraints.

Currently, a full table scan is performed whenever RUNSTATS on TABLE is run. With data sampling, only a subset of the data is scanned.

If a query attempts to project overall trends and patterns, and approximate answers within a certain margin of error may be sufficient to detect these trends and patterns, then data sampling may be a better alternative than a full table scan.

In DB2 V8.1, the SAMPLED DETAILED CLAUSE was introduced to allow sampling for computing detailed index statistics. Using this clause reduces the amount of background calculation performed for detailed index statistics and the time required, but produces adequate precision in most cases.

Here are some examples of its usage:

**Example 27. Collect detailed catalog statistics on both indexes, but use sampling instead of performing detailed calculations for each index entry**

```
RUNSTATS ON TABLE db2admin.department AND SAMPLED DETAILED INDEXES ALL
```

**Example 28. Collect detailed sampled statistics on indexes as well as distribution statistics for the table**

```
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION ON KEY
   COLUMNS AND SAMPLED DETAILED INDEXES ALL
```
In DB2 V8.2, two new methods of table data sampling are offered: row-level Bernoulli sampling and system page-level sampling.

**RUNSTATS with row-level Bernoulli sampling**

Row-level Bernoulli sampling gets a sample of P percent of the table rows by means of a sargable (search + argument-able predicate is a predicate that can be evaluated by the Data Manager) predicate that includes every row in the sample with the probability of P/100 and excludes it with a probability of 1 - P/100.

For example, for 10% Bernoulli sampling, (10/100) 10% of the rows will be selected and (1-10/100) 90% of the rows will be rejected. In Bernoulli sampling, an I/O will be incurred for each page since the table will be scanned. A random number will be generated to determine if a row will be selected or not (similar to flipping a coin with a probability of P/100). Even if the I/O happens for every page, we still save on CPU time required to process the data.

With row-level Bernoulli sampling, every data page is read. However, it can still produce significant performance improvement because RUNSTATS is CPU intensive. If indexes are available, then the sampling is improved. It may also provide more accurate statistics if the data is clustered (obtains a sample that better represents overall table data).

**RUNSTATS with system page-level sampling**

System page-level sampling is similar to row-level sampling, except that the pages are sampled and not rows. Each page is selected with a probability of P/100 and rejected with a probability of 1 - P/100. For each page selected all of the rows are selected. The benefit of system page-level sampling over a full table scan or Bernoulli sampling is the savings in I/O.

The sampled pages are also prefetched, so this method would be faster than row-level Bernoulli sampling. Compared with no sampling, page-level sampling significantly improves performance.

The RUNSTATS repeatable clause allows the same sample to be generated across RUNSTATS statements a long as the table data has not changed. To specify this option, the user must also supply an integer that will represent the seed to be used for sample generation. By using the same seed, the same sample can be generated.

In summary, the accuracy of statistics depends on the sampling rate, the data skew, and data clustering for data sampling.

Some examples of RUNSTATS using Bernoulli row-level and system page-level sampling are as follows:

**Example 29. Collect statistics, including distribution statistics on 10 percent of the rows**

```sql
RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION TABLESAMPLE BERNOULLI (10)
```
Example 30. To control the sample set on which the statistics will be collected and to be able to repeatedly use the same sample set, use the following:

RUNSTATS ON TABLE db2admin.department WITH DISTRIBUTION TABLESAMPLE BERNOULLI (10) REPEATABLE (1024)

Example 31. To collect index statistics as well as table statistics on 10 percent of the data pages. Note that only the table data pages and not the index pages are sampled. In this example, 10 percent of table data pages are used for the collection of table statistics, while for index statistics all of the index pages will be used.

RUNSTATS ON TABLE db2admin.department AND INDEXES ALL TABLESAMPLE SYSTEM (10)

Alternative ways of collecting catalog statistics

One alternative way of collecting statistics for all tables in a database is to issue a REORGCHK command as follows:

Example 32. Collect catalog statistics for all tables using REORGCHK

REORGCHK UPDATE STATISTICS ON TABLE ALL

The update statistics option of the reorgchk command is similar to invoking the RUNSTATS routine to update catalog statistics for all tables in the database, but statistics on all columns is gathered using the default RUNSTATS options only. Using this command, it is also possible to collect statistics on individual tables or a group of tables with the same schema.

Example 33. Collect catalog statistics for a table using REORGCHK

REORGCHK UPDATE STATISTICS ON TABLE db2admin.department

Example 34. Collect catalog statistics for a schema using REORGCHK

REORGCHK UPDATE STATISTICS ON SCHEMA syntools

Although this method seems to be a quick way of collecting catalog statistics on multiple tables, it is advisable only for situations where the REORGCHK execution can be done in a reasonable amount of time.

Collecting catalog statistics can also be issued during a LOAD REPLACE of a table, as well as during the creation of an index.

In UDB V8.2, to invoke the collection of catalog statistics during a LOAD, the option "STATISTICS USE PROFILE" must be added to the LOAD control statements. The option "STATISTICS YES" will still work, but is now obsolete syntax. The goal of issuing a "STATISTICS USE PROFILE" during a LOAD is to gain savings in elapsed time over the usual process where a LOAD of a table
is followed by a RUNSTATS on that same table. The statistics profile must be created before the load is executed, and it now allows one to specify the same statistics options as one would in a RUNSTATS command.

Example 35. Collect catalog statistics using the LOAD command and the STATISTICS USE PROFILE clause

```
LOAD FROM inputfile.del OF DEL REPLACE INTO db2admin.department STATISTICS USE PROFILE
```

In most cases, a RUNSTATS is done after an index has been created. However, it is possible to collect index statistics while a create index operation is being done. This avoids another scan of the index in order to gather statistics. The following examples illustrate some of the options available to do this:

Example 36. To collect basic index catalog statistics:

```
CREATE INDEX db2admin.inx1 ON db2admin.department (deptno) COLLECT STATISTICS
```

Example 37. To collect extended index catalog statistics:

```
CREATE INDEX db2admin.inx2 ON db2admin.department (deptname) COLLECT DETAILED STATISTICS
```

Example 38. To collect extended index catalog statistics specifying that sampling should be used:

```
CREATE INDEX db2admin.inx3 ON db2admin.department (deptname) COLLECT SAMPLED DETAILED STATISTICS
```

RUNSTATS authorization and user access

To be able to collect statistics against a table or index, one must be able to 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

There are two options for user access to a table while a RUNSTATS is running against that table: allow read access and allow write access.

The parameter ALLOW READ ACCESS in a RUNSTATS command specifies that other users can have read-only access to the table while statistics are calculated.

The parameter ALLOW WRITE ACCESS in a RUNSTATS command specifies that other users can read from and write to the table while statistics are calculated. Use this clause if the table must be up and available at any time.
RUNSTATS in partitioned databases

When a RUNSTATS command is issued in a partitioned database, and a table partition exists in the database partition where the RUNSTATS was issued, RUNSTATS executes on that database partition. If a table partition does not exist on the database partition, the request is sent to the first database partition in the database partition group that holds a partition for that table. RUNSTATS then executes on that database partition.

RUNSTATS does not run in parallel in a partitioned database, but information is determined for one partition, and then extrapolated to determine a suitable estimate for all partitions. There is an implicit assumption that the rows of each table are evenly distributed across all the partitions in each multiple-partition database partition group.

The following IBM technote talks about a possible problem that may be encountered with RUNSTATS when loading a table. The resolution is to ensure that a partitioning key is defined for the table before a load takes place.

IBM Technote # 1153232 (2004-02-12):
"Runstats updates column CARD in syscat.tables to zero rows on multi partitioned instance after load.

Abstract

If one of the partitions has zero rows in it for a given table following a load and runstats is executed from that partition, it will update the column CARD in syscat.tables to show that the table has zero rows. This only affects multi-partition instances.

Content

The behavior of runstats is such that it will use the data on the partition that it is run from to extrapolate the number of rows in the table. For example, if after a load all the data is on one partition and runstats is executed from another partition, it will update syscat.tables to show zero rows for that table. However, if runstats is run from the partition that has all the data, it will update syscat.tables to show the table having ((num rows on that partition) x (number of partitions)) rows.

To get around this issue, which is the default behavior of runstats, a partitioning key needs to be defined for the table before the load takes place. The partitioning key needs to be on a column or set of columns that are highly unique thus ensuring that the data will be evenly distributed across all the partitions that the table is defined on."

Adjusting the STAT_HEAP_SZ database configuration parameter

The stat_heap_sz or the statistics heap size database configuration parameter indicates the maximum size of the heap used in collecting statistics using the RUNSTATS command. It is allocated when the RUNSTATS utility is started, then freed when it completes. Stat_heap_sz is part of agent private memory. When gathering distribution statistics, it is recommended to increase the stat_heap_sz parameter so that more columns can fit into the heap. Wide tables would also
require more memory for processing. When executing RUNSTATS with the SAMPLED DETAILED option, an additional 2 Megabytes of memory must be allocated to ensure that RUNSTATS can run successfully.

**Strategies to reduce RUNSTATS impact on system performance**

You can use the following strategies to help reduce RUNSTATS performance impact on the system:

- Run RUNSTATS on only a few tables and indexes at a time, rotating through the set of tables.
- Specify only those columns for which data distribution statistics should be collected. Specify only those columns that are used in predicates.
- Implement multiple concurrent RUNSTATS on different partitions for different tables.
- Perform RUNSTATS only on those critical tables that would have an effect on the current workload. Avoid running RUNSTATS on tables that don't need it.
- Adjust the frequency of RUNSTATS depending on how fast data is changing in a table.
- Adjust the frequency and detail of RUNSTATS depending on how fast it takes to complete the run for the table.
- Schedule RUNSTATS to run only during low-activity periods in the system.
- Throttle RUNSTATS to minimize its demands on the system.

Scheduling RUNSTATS to run only during low-activity periods in the system is a good way of minimizing the impact to the system. However, for a 24 x 7 system, there may be no available window or low-activity period in the system. One way to handle this situation is to use the RUNSTATS throttling option.

Throttling limits the amount of resources taken up by a utility based on the current database activity levels. In UDB, the interaction of the util_impact_lim and the UTIL_IMPACT_PRIORITY parameters determine the behavior of RUNSTATS when throttled. The UTIL_IMPACT_PRIORITY keyword is used in a clause in a utility command such as RUNSTATS, while util_impact_lim is an instance configuration parameter.

The util_impact_lim parameter is the percentage of allowable impact that all throttled utilities can have on the instance workload. If the util_impact_lim is 100 (the default value), no utility invocations such as RUNSTATS are throttled. If for example, the util_impact_lim is set to 10, then a throttled RUNSTATS invocation is limited to consume not more than 10 percent of the workload.

The UTIL_IMPACT_PRIORITY keyword serves as a switch to specify whether RUNSTATS subscribes to a throttling policy or not.

**Example 39. Use of UTIL_IMPACT_PRIORITY keyword**

```
RUNSTATS ON TABLE db2admin.department AND INDEXES ALL UTIL_IMPACT_PRIORITY 10
```

where 10 is the priority or the level at which RUNSTATS will be throttled.

When util_impact_lim is not 100, and RUNSTATS is invoked with UTIL_IMPACT_PRIORITY, it will be throttled. If priority is not specified for UTIL_IMPACT_PRIORITY and util_impact_lim is not 100,
RUNSTATS will have a default priority of 50 and will be throttled. If the UTIL_IMPACT_PRIORITY keyword is omitted from the RUNSTATS command, RUNSTATS will run unthrottled. If a priority is specified, but util_impact_limit is set to 100, RUNSTATS will run unthrottled. If priority is set to zero, RUNSTATS will run unthrottled.

When RUNSTATS throttling is defined and operational, the RUNSTATS utility is expected to take longer, but will pose a lesser impact to the system.

**Miscellaneous RUNSTATS issues**

**RUNSTATS on declared global temporary tables**

The RUNSTATS command can be issued on a declared global temporary table (DGTT), but the resulting statistics are not stored in the system catalogs because declared temporary table entries do not have catalog entries. These statistics are stored in memory structures that represent the catalog information for declared temporary tables. The recommendation is to issue RUNSTATS if the savings in total query elapsed time is much greater than the additional time to do RUNSTATS.

**Can the RUNSTATS command be issued from within an SQL routine (procedure, UDF, method)?**

The following IBM technote discusses the question:

**IBM Technote # 1153051**

"The RUNSTATS command cannot be issued from an SQL routine (procedure, UDF, or method), an SQL dynamic compound statement, nor can it be issued using an operating system call issued from within an SQL routine. If you need to issue the RUNSTATS command, issue it directly from the Command Line Processor (CLP) directly or from a client application or external language routine by using the db2Runstats() API function (as of DB2 8.1), as well as through LOAD and CREATE INDEX statements."

It is possible to create a C stored procedure named "runstats" that invokes the db2Runstats() API function. This C stored procedure can then be invoked using the CALL command.

**How do I perform a RUNSTATS to gather statistics from the call level interface (CLI) or Java?**

The following IBM technote answers the question:

**IBM Technote # 1023788:**

"The DB2 administrative API (programming interface) sqlustat can be used to issue a RUNSTATS from a DB2 application, such as one written to the Call Level Interface. sqlustat is described in the DB2 Administrative API Reference manual."

**DB2 automatic statistics collection**

DB2 automatic statistics collection was introduced in DB2 UDB Version 8.2. Automatic statistics collection is part of a completely automated table maintenance solution. The objective is to
allow DB2 to determine which statistics are required by the workload, and automatically run the RUNSTATS utility in the background periodically to update statistics as needed.

To set up the SAMPLE database for automatic statistics collection, set the database configuration parameters for automatic maintenance switches as follows:

```
db2 update db cfg for SAMPLE using AUTO_MAINT ON
db2 update db cfg for SAMPLE using AUTO_TBL_MAINT ON
db2 update db cfg for SAMPLE using AUTO_RUNSTATS ON
```

The following diagram (only showing the statistics collection and profiling options) shows the hierarchy of automatic maintenance commands for statistics collection and profiling and the dependencies. With this structure, it is possible to quickly turn off the parent automatic maintenance parameter, AUTO_MAINT, at the highest level without losing the configuration settings for the lower levels, i.e. AUTO_RUNSTATS.

**Figure 1. Automatic Maintenance Settings**

```
 Automatic maintenance           (AUTO_MAINT)  
       ↓                           ↓          
 Automatic table maintenance     (AUTO_TBL_MAINT)  
          ↓                       ↓           
 Automatic runstats              (AUTO_RUNSTATS)  
   Automatic statistics profiling (AUTO_STATS_PROF)  
                   ↓                     ↓          
 Automatic profile updates       (AUTO_PROF_UPD)  
```

The parameters AUTO_STATS_PROF and AUTO_PROF_UPD are turned on if automatic statistics profiling is desired. Automatic statistics profiling goes one step further by determining when and how to collect statistics. A statistics profile is generated automatically for use by the automatic statistics collection process to schedule RUNSTATS. An internal algorithm is used to compare newly collected statistics with a saved set of statistics, and RUNSTATS with sampling is issued based on certain conditions being triggered.

When automatic statistics profiling is enabled, information about database activity is collected and stored in a query feedback warehouse. Statistical profiles are then generated based on the data in the query feedback warehouse.

To enable automatic statistics profile generation, set two database configuration parameters on:

- db2 update db cfg for SAMPLE using AUTO_STATS_PROF ON

Turning this parameter on activates the collection of query feedback data.

- db2 update db cfg for SAMPLE using AUTO_PROF_UPD ON

Turning this parameter on specifies that the RUNSTATS profile is to be updated with recommendations from DB2 analyzing the query feedback data.
Before automatic statistics profiling is activated, a query feedback warehouse must be created by running the SYSINSTALLOBJECTS stored procedure.

This stored procedure is invoked as follows:

```sql
call SYSINSTALLOBJECTS ( toolname , action , tablespacename , schemaname )
```

where toolname is "ASP" or "AUTO STATS PROFILING" and action is "C" for create, and "D" for drop.

The parameter schemaname is not currently used.

For example, to create the feedback warehouse, run the followed stored procedure:

```sql
call SYSINSTALLOBJECTS ('ASP', 'C', 'USERSPACE1', '')
```

This procedure will create the following tables that store recommendations, and information about the predicates encountered during query execution:

- SYSTOOLS.OPT_FEEDBACK_QUERY
- SYSTOOLS.OPT_FEEDBACK_PREDICATE
- SYSTOOLS.OPT_FEEDBACK_PREDICATE_COLUMN
- SYSTOOLS.OPT_FEEDBACK_RANKING
- SYSTOOLS.OPT_FEEDBACK_RANKING_COLUMN

When the AUTO_PROF_UPD parameter is disabled, recommendations are stored in the SYSTOOLS.OPT_FEEDBACK_RANKING table. The recommendations stored in this table can then be reviewed and refined when manually updating the RUNSTATS profile.

DB2 automatic statistics collection and profile generation can only be activated in DB2 serial mode; they are not available in federated, SMP, and MPP environments.

**Conclusion**

Understanding and properly using RUNSTATS is one of the keys to maintaining a high-performing database. In this article, I've discussed the importance of RUNSTATS and explained the many options you can use to make it more useful, including new options in V8.2 which ease administration by letting you set up a profile specifying which statistics you want gathered for each table. The examples I've shown will make it easy for you to use RUNSTATS to gain optimal performance for your own databases.
Related topics

- The DB2 Information Center contains more details on RUNSTATS and analyzing statistics.
- The DB2 UDB V8.2 Command Reference contains the details for the RUNSTATS command syntax.

© Copyright IBM Corporation 2004
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)