DB2 for Linux, UNIX and Windows has supported automatic statistics collection since V8.2. However, even automatic statistics collection may not collect statistics soon enough for certain workloads. V9.5 adds the capability to collect statistics when the query optimizer needs them - at statement compilation time. DB2 9.5 for Linux, UNIX and Windows supports a number of low overhead methods to fabricate or collect statistics while a statement is being compiled. Statistics are immediately made available to other queries through a new statistics cache. Background statistics collection is improved so that it occurs sooner than before, to ensure your tables always have the most current statistics. A number of new monitoring, logging and Explain features have been added to help you understand all forms of DB2 statistics collection, whether it is done manually by you with the RUNSTATS command or automatically by DB2.
Agenda

• Real-time statistics (RTS) overview
• Synchronous statistics collection
• Statistics fabrication
• Asynchronous statistics collection
• Statistics caching
• Statistics activity logging and monitoring
• Observing real-time statistics with the explain facility
Real-time Statistics Overview

- Collect statistics at statement compilation time
  - Table(s) identified based on statistical needs of the query and amount of data change
  - Statistics can either be collected with RUNSTATS or fabricated
  - Statistics are immediately made available to other connections via a new statistics cache
  - Catalogs are not updated immediately to minimize overhead
- Request immediate background (asynchronous) statistics collection
- Why are RTS important?
  - Because accurate statistics are essential for query optimization
RUNSTATS review

- Utility to gather statistics on tables and indexes
- Statistics are essential for query optimization
  - Used to compute access plan cost and cardinality
- Physical statistics
  - E.g. Number of pages in table, number of levels in an index
- Data statistics
  - E.g. Number of rows in table, number of distinct values in a column
- Statistics are stored in the system catalogs
  - SYSCAT.TABLES, SYSCAT.COLUMNS, SYSCAT.INDEXES, SYSCAT.COLDIST etc.

When the SQL compiler optimizes SQL query plans, its decisions are heavily influenced by statistical information about the size of the database tables and indexes. The optimizer also uses information about the distribution of data in specific columns of tables and indexes 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. 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. Statistical information is collected for specific tables and indexes in the local database when you execute the RUNSTATS utility. Statistics are collected only for the table partition that resides on the partition where you execute the utility or the first partition in the database partition group that contains the table. The collected statistics are stored in the system catalog tables.
Comparison to DB2 for z/OS RTS

- Real-time statistics refers to any statistic representing data server operation, collected in real time
- DB2 for z/OS:
  - Continually collects RTS in memory
  - Persists RTS to 2 tables in a special DB (DSNRTSDB) periodically, or when the table space, DB or server is stopped
- DB2 9.5 for Linux, UNIX and Windows
  - Continually collects RTS in memory
  - Periodically persisted to disk, but not all RTS are externalized
  - RTS are used to derive certain table and index statistics
  - RUNSTATS frequency is determined by query workload and amount of data change indicated by RTS
  - RTS are used by the query optimizer
- Data server functionality will converge in future releases


Some table and table space statistics can be determined from the DB2 for LUW system monitor.
RTS and Automatic Statistics Collection

- DB2 supports automatic statistics collection since V8.2
- Statistics are collected in the background (asynchronously) via RUNSTATS
- Tables are identified by amount of data change over time
- Asynchronous collection is unobtrusive
  - Throttled (maximum 7% overhead)
  - Non-blocking using low-priority locks
- Table evaluation occurs every 2 hours
  - This may be too infrequent for some applications
- What tables to collect and when (maintenance window) can be controlled via the health monitor

See the following link for a detailed article on how the existing automatic statistics collection works.

Real-time Statistics Activation

- DB configuration parameter (AUTO_STMT_STATS)
  - Default is OFF for both new and migrated DBs
- Under automatic table maintenance hierarchy
  - AUTO_RUNSTATS can be ON while AUTO_STMT_STATS is OFF
  - AUTO_STMT_STATS can't be ON unless AUTO_RUNSTATS is ON

<table>
<thead>
<tr>
<th>Automatic maintenance</th>
<th>(AUTO_MAINT) = ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic database backup</td>
<td>(AUTO_DB_BACKUP) = OFF</td>
</tr>
<tr>
<td>Automatic table maintenance</td>
<td>(AUTO_TBL_MAINT) = ON</td>
</tr>
<tr>
<td>Automatic runstats</td>
<td>(AUTO_RUNSTATS) = ON</td>
</tr>
<tr>
<td><strong>Automatic statement statistics</strong> (AUTO_STMT_STATS) = ON</td>
<td></td>
</tr>
<tr>
<td>Automatic statistics profiling</td>
<td>(AUTO_STATS_PROF) = OFF</td>
</tr>
<tr>
<td>Automatic profile updates</td>
<td>(AUTO_PROF_UPD) = OFF</td>
</tr>
<tr>
<td>Automatic reorganization</td>
<td>(AUTO_REORG) = OFF</td>
</tr>
</tbody>
</table>
Configuring RTS Time Limit

• Use optimization profiles to configure an RTS time limit
• New RTS request element
• Can also disable RTS for selected statements or groups of statements
• Time is specified in milliseconds
  • Set the RTS time limit to 3.5s
    • <RTS TIME="3500" />
  • Disable RTS for a particular statement
    • <RTS OPTION="DISABLE" />
• Can’t enable RTS with optimization profile unless RTS is enabled for database e.g. AUTO_RUNSTATS and AUTO_STMT_STATS must be ON
• Can be specified as global or statement-level request


RTS requests
The RTS general request element can be used to enable or disable real-time statistics collection. It can also be used to limit the amount of time taken by real-time statistics collection. For certain queries or workloads, it may be desirable to disable or limit the time spent on real-time statistics collection to avoid extra overhead at statement compilation time.

Description
The RTS general request element has two optional attributes.
The OPTION attribute is used to enable or disable real-time statistics collection. It can take the values ENABLE or DISABLE. ENABLE is the default if no option is specified.
The TIME attribute specifies the maximum amount of time in milliseconds to be spent on real-time statistics collection at statement compilation time, for a single statement.
If ENABLE is specified for the OPTION attribute, automatic statistics collection and real-time statistics must be enabled by their corresponding configuration parameters. Otherwise, the optimization guideline will not be applied, and you will get warning message SQL0437W (reason code 13).
For example, the following RTS request enables real-time statistics collection and limits real-time statistics collection time to 3.5 seconds.
<RTS OPTION="ENABLE" TIME="3500" />
Real-time Statistics Deep Dive

• RTS is an autonomic feature
• You don’t need to know the implementation details!
• But it might be fun to peek under the hood...
  • Synchronous statistics collection
  • Statistics fabrication
  • Asynchronous statistics collection
  • Statistics caching
  • Statistics activity logging and monitoring
  • Observing real-time statistics with the explain facility
The box on the left represents the existing Automatic Statistics Collection process. This process runs asynchronously to DB operations such as SQL statement execution. The statistics evaluator determines if statistics should be collected and then initiates a ‘background’ RUNSTATS. The collected statistics are stored in the system catalogs. The statistics for a particular table are loaded from the system catalogs into a catalog cache, whenever they are referenced by the SQL Compiler.
V9.5 extends the current architecture indicated by the green boxes. These boxes will be described in more detail on subsequent pages.

**Statistics Cache** – Synchronously collected statistics or fabricated statistics are stored here, until they are written to the system catalogs. The Catalog Cache loads new statistics from here, if they have not yet been written to the system catalogs.

**Sensitivity Analyzer** – This new component of the optimizer determines whether a statement requires statistics, what statistics it requires and how to collect them.

**Synchronous Collector** – This new component collects statistics synchronously (as part of SQL statement compilation) by either invoking RUNSTATS or by fabricating statistics.

**Asynchronous Request Queue** – If the Sensitivity Analyzer determines that statistics should be collected asynchronously, a request is added to this queue. Every 5 minutes, a background process will be invoked to process requests in this queue.
Sensitivity Analyzer

- Drives synchronous statistics collection by determining:
  1. If the query needs statistics
  2. Which tables referenced by the query require updated statistics
  3. How to collect the statistics
One of the goals of the sensitivity analyzer is to minimize the amount of synchronous statistics collection for OLTP applications. The overhead of synchronous collection may be more apparent to OLTP applications, however they may be able to tolerate less accurate statistics.
Sensitivity Analyzer

- Determining what tables require updated statistics
  - Determine if statistics are stale
    - Based on amount of data change and table size
    - Data change indicated by row count of update, delete and insert activity (UDI counter)
    - Smaller tables require a larger %age UDI change, larger tables require a smaller %age UDI change
  - Algorithm is consistent with automatic statistics collection
Sensitivity Analyzer

• Determine missing ‘interesting’ statistics
  • If any interesting statistics are missing, statistics are collected regardless of amount of data change
  • Interesting statistics are determined by how columns are used in the query
    • “WHERE NAME = ?” can’t use distribution statistics
      (Unless REOPT ONCE/ALWAYS is specified)
    • “WHERE NAME = ‘Jones’” can use distribution statistics
  • Considers options specified in the statistics profile
    • E.g. Predicate may be able to use distribution statistics but they aren’t specified in the statistics profile

More details on statistical profiles and RTS are provided later in this presentation. Repeating an overview here:

The RUNSTATS utility provides a statistical profile facility to:
1) register a statistical profile, while optionally gathering statistics
2) modify an existing statistical profile stored in the catalogs, while optionally gathering statistics
3) repeatedly gather statistics on the table using an already registered statistics profile for that particular table.

This may be convenient for multiple scripts that need to perform RUNSTATS on the same set of tables so the RUNSTATS options don’t need to be repeated in every script. Additionally, the statistical profile can be specified for LOAD so that consistent RUNSTATS options can be used and don’t need to be repeated on the LOAD command.

When a statistical profile is registered, a RUNSTATS command string corresponding to that profile is at the same time built and stored in the STATISTICS_PROFILE column of the catalog table SYSIBM.SYSTABLES. An internal version of the profile is also maintained in the system catalogs in SYSTABLES.PACKED_DESC.
Sensitivity Analyzer – Statistics Collection Methods

- Determine how statistics should be collected
- Based on UDI and interesting statistics
- Methods available:
  - Fabrication
    - Derive subset of statistics from index and data manager metadata
    - i.e. Internal real-time statistics
    - Very fast
  - Synchronous collection
    - Perform RUNSTATS to collect full statistics
    - Within a time budget
  - Asynchronous collection
    - Schedule background RUNSTATS collection
    - Same mechanism as automatic statistics collection
Sensitivity Analyzer – Statistics
Collection Methods

• Fabrication
  • Partial fabrication to update HIGH2KEY/LOW2KEY, adjust histograms* if necessary
    • Small data change, range predicate on column, column leading in an index
      • E.g. ORDER_DATE > ’01/31/2008’
    • Allows optimizer to quickly see new values introduced into range
  • Full fabrication for table and index statistics
    • Data change didn’t warrant full synchronous collection but statistics are slightly stale OR
      • Previous synchronous collection timed out

* Also known as quantiles. Stored in SYSSTAT.COLDIST catalog.

Some applications issue queries with search conditions looking for newly inserted data. However, if the statistics aren’t current, they may not include the newly inserted data. For example, the HIGH2KEY statistic for the ORDER_DATE column may be ’01/15/2008’ but the query is looking for newer order dates. Consequently, the optimizer will assume few rows qualify. Partial statistics fabrication makes the optimizer aware of the newer order dates, providing there is an index that includes ORDER_DATE as a leading column. The index will be quickly probed to determine the second highest and lowest values, which are then used to update the HIGH2KEY and LOW2KEY statistics. Any existing histogram statistics will be adjusted too. Partial fabrication occurs more frequently than full fabrication or synchronous collection, however it has lower overhead. More details on full fabrication are on the next page.
Partial statistics fabrication

WHERE ORDER_DATE > '01/31/2008'

HIGH2KEY = 02/01/2008 -> 03/01/2008

VALCOUNT is the number of rows whose value is less than or equal to COLVALUE
Sensitivity Analyzer – Statistics Collection Methods

• Synchronous collection
  • Orders tables smallest to largest
  • Uses RUNSTATS sampling for large tables
    • For large tables, fabricate statistics for indexes with existing statistics
    • RUNSTATS sampling not supported for indexes

• Asynchronous collection
  • Performed within 5 minutes
  • Asynchronous collection is always scheduled:
    • When statistics are sampled or fully fabricated
      • No asynch for partial fabrication – could drive too many asynch requests
    • Statistics could be inferior – get full statistics ASAP

Synchronous statistics collection will sample tables that have more than 4000 pages. This value and algorithm may change in the future. Synchronous collection orders the tables smallest to largest in order to prevent small tables from being starved by larger tables.

Asynchronous collection is always scheduled if the statistics are sampled or fully fabricated, to ensure the most accurate statistics are stored in the system catalogs. The in-memory RTS are intended to provide the query optimizer with something more accurate than the stale statistics, however the statistics stored in the system catalogs should correspond to the options specified in the statistical profile, whether it is the default profile or a profile provided by the user.
Statistics Fabrication

- Derive statistics by:
  - Probing high and low end of index to get HIGH2KEY and LOW2KEY
    - Adjust histograms, if necessary
  - Using statistics dynamically maintained by the index manager
    - FULLKEYCARD
    - Also used for column cardinality (COLCARD) when provided by a single column index
    - NLEAF
    - NLEVELS
  - Using statistics dynamically maintained by the data manager
    - FPAGES
    - Derive table cardinality, based on lower of value derived from:
      - Page size and avg. row width OR
      - Individual counts of rows inserted, updated, deleted
    - Derive # of active blocks based on extent size (for MDC tables)
    - Extrapolate some other statistics (if they exist)
      - Column cardinality, column group cardinality
      - Adjust existing statistics to ensure consistency

There are a number of ways that statistics can be fabricated:

- The index can be probed to get the 2nd highest and lowest values in order to fabricate HIGH2KEY and LOW2KEY

- The Index Manager component maintains the full key cardinality (FULLKEYCARD), the number of leaf pages (NLEAF) and the number levels (NLEVELS). These statistics can be used directly by the optimizer.

- The Data Manager component maintains the number of file pages (FPAGES). This value can be used directly by the optimizer. The table’s cardinality can be derived from FPAGES because the page size and the average row width are known. Also, the Data Manager maintains individual counts for the number or rows inserted, updated and deleted. These counts can also be used to compute the table cardinality. Statistics fabrication uses the lower value of these two methods.

- The number of active blocks for multi-dimensional clustered (MDC) tables can be derived knowing the extent size and FPAGES.

- Other statistics that can’t be derived can be extrapolated from existing statistics.

The statistics fabrication process will ensure consistency between all derived and extrapolated statistics.
Sensitivity Analyzer – Other Considerations

- VOLATILE tables are not considered
  - Statistics could be changing too frequently
  - Drives too many statistics collection
  - Existing fabrication and heuristics used for optimization
- Asynchronous collection is not done for DGTTs
  - No catalog entries to retain statistics
  - Current connection needs immediate statistics (synchronous or fabricated)
- Synchronous collection is not done if DGTT already has statistics
  - Minimize dynamic statement cache invalidation
    - Remember that static SQL referencing DGTTs uses incremental bind e.g. essentially dynamic
Sensitivity Analyzer – Other Considerations

- Tables with manually updated statistics are not considered for either synchronous or asynchronous collection
  - i.e. UPDATE SYSSTAT.TABLES SET CARD = 500 ...
  - User has assumed responsibility for maintaining statistics manually
  - Would be bad for db2look simulations!
  - An explicit RUNSTATS will re-enable for consideration
- Table truncated via IMPORT is considered to have no statistics
- Synchronous collection/fabrication not performed for SET INTEGRITY or REFRESH TABLE
  - Hybrid DDL and DML statements
Statistics Cache

- Synchronous and fabricated statistics are not stored in the system catalogs
  - Requires considerable I/O,
  - Could cause lock contention
- Stored in a statistics cache instead
- Written to system catalogs asynchronously, soon after collection
  - Typically, within 5 minutes
- Synchronous and fabricated statistics are available to other compilation requests once they are stored in statistics cache
  - Do need to wait to become available in system catalogs
- Statistics cache is part of existing catalog cache
- Only exists on catalog DB partition in a DPF environment
- Statistics cache contents can be displayed with db2pd tool

A statistics cache was introduced in DB2 9.5 to make synchronously-collected statistics available to all queries. This cache is part of the catalog cache. In a partitioned database environment, this cache resides only on the catalog database partition. The catalog cache can store multiple entries for the same SYSTABLES object, which increases the size of the catalog cache on all database partitions. Consider increasing the value of the catalogcache_sz database configuration parameter when real-time statistics collection is enabled
Statistics Cache

```
db2pd -alldbs -statisticscache details

Database Partition 0 -- Database PROD1 -- Active -- Up 10 days 02:56:13
Statistics Cache: 330264
High Water Mark: 458752
Entries in Statistics Cache:
Address Schema Name LastRefID LastStatsTime
0x074520303CDB2USER PRODUCTS 1801 2007-05-03-11.53.04.104073 V

--- TABLE PACKED DESCRIPTOR: -----
PACKED DESCRIPTOR HEADER
No. of rows : 369
No. of pages for table : 3
No. of overflow records : 0
No. of indexes : 0
No. of xml indexes : 0
Total no. of columns : 2
MDC : NO
DGTT options : 0
Avg Row Compression Ratio : 0.000000
Percentage of rows compressed : 0.000000
Avg length of compressed row : 0
Avg row size : 20
Active Blocks : 0

COLUMN DESCRIPTION
COLUMN NAME : C1
Column id : 0
No. unique values in col. : 128
```

This page shows an example of the db2pd tool output displaying the contents of an entry in the statistics cache.
RTS and Catalog Cache

- Synchronous statistics collection doesn’t ‘hard’ invalidate existing entries in the catalog cache
  - Requires waiting for the connection using the entry to release it
- Existing entries are ‘soft’ invalidated
  - Marked invalid, but existing connection can continue to use them
  - Flushed once the connection releases them
- New catalog lookups load the latest entry from the statistics cache
  - If the statistics haven’t already been ‘hardened’ to disk
- Consequently, the catalog cache could contain multiple entries for the same table
  - N are marked soft invalid
  - 1 is valid (current)
- db2pd can be used to view the catalog cache
- Consider increasing the value of the catalogcache_sz database configuration parameter
There are 2 entries for table DB2USER.ORDER_LINE in the catalog cache. The status (Sts) for the first one is V=valid. The status for the second is S=soft invalid. The second entry was soft invalidated because synchronous statistics were collected or fabricated while another DB connection was using the entry. Subsequent references to DB2USER.ORDER_LINE will use the new valid entry and the soft-invalid entry will be flushed from the cache after the current connection releases it.

**SYSTABLES:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Schema</th>
<th>Name</th>
<th>Type</th>
<th>TableID</th>
<th>TbspaceID</th>
<th>LastRefID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x780000232FF820</td>
<td>SYSIBM</td>
<td>SYSTABLES</td>
<td>T</td>
<td>5</td>
<td>0</td>
<td>19288214</td>
</tr>
<tr>
<td>0x0001000078000000232FF820</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x780000232FD360</td>
<td>SYSCAT</td>
<td>TABLES</td>
<td>V</td>
<td>0</td>
<td>19288214</td>
<td></td>
</tr>
<tr>
<td>0x00010000780000232FD360</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x780000232FCF40</td>
<td>DB2USER</td>
<td>CUSTOMER</td>
<td>0</td>
<td>0</td>
<td>19288214</td>
<td></td>
</tr>
<tr>
<td>0x00010000780000232FCF40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x780000232FCF40</td>
<td>DB2USER</td>
<td>ORDER_LINE</td>
<td>T</td>
<td>4</td>
<td>2</td>
<td>19288214</td>
</tr>
<tr>
<td>0x00010000780000232FCF40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x780000232FCF40</td>
<td>DB2USER</td>
<td>ORDER_LINE</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>19288214</td>
</tr>
<tr>
<td>0x00010000780000232FCF40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x780000234433A0</td>
<td>DB2USER</td>
<td>DISTRICT</td>
<td>0</td>
<td>0</td>
<td>19288214</td>
<td></td>
</tr>
<tr>
<td>0x00010000780000234433A0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x780000234433A0</td>
<td>DB2USER</td>
<td>ORDER_LINE</td>
<td>0</td>
<td>0</td>
<td>19288214</td>
<td></td>
</tr>
<tr>
<td>0x00010000780000234433A0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Address:** Address of catalog cache entry.
- **Schema:** The schema qualifier for the table.
- **Name:** The name of the table.
- **Type:** The type of the table.
- **TableID:** The table identifier.
- **TbspaceID:** The identifier of the table space where the table resides.
- **LastRefID:** The last process identifier that referenced the table.
- **CatalogCache LoadingLock:** The name of the catalog cache loading lock for the cache entry. A lock is acquired when the catalog cache entry is being loaded.
- **CatalogCache UsageLock:** The name of the usage lock for the cache entry. A lock is acquired when the catalog cache entry is being referenced.
- **Sts:** The status of the entry. The possible values are:
  - V (valid).
  - I (invalid).
  - S (soft invalid. Catalog cache entries become soft invalid when statistics have been updated by real-time statistics collection. These catalog cache entries may still be used by a database agent, but they are not valid for use by a new catalog cache request. Once the soft invalid entry is no longer in use, it will be removed. New catalog cache requests will use the valid entry.)
Static and Dynamic SQL Statements

• Dynamic statements
  • Synchronous and asynchronous collection invalidates cached dependent dynamic statements
    • So they can be recompiled with the most current statistics
  • Dynamic statement cache doesn’t perform sensitivity analysis
    • Minimize overhead for cache lookup
    • High dynamic statement cache hit rate means few synchronous requests
    • Regular automatic statistics collection may cause periodic invalidation
  
• Static statements
  • Static packages are not invalidated by synchronous or asynchronous collection
    • Must perform manual BIND/REBIND as today
  • RTS can occur during bind for static statements
Careful consideration was given to minimize the overhead of Automatic Statistics Collection.
RTS and Health Monitor Policy

- Health monitor policy controls automatic table maintenance activities
  - Maintenance window (when to collect statistics)
  - Table identification (tables for which statistics should be collected)
  - Specified using:
    - Control Centre
    - SYSPROC.AUTOMAINT_SET_POLICY stored procedure
    - SYSPROC.AUTOMAINT_SET_POLICYFILE stored procedure
- Maintenance window doesn't apply to synchronous collection
  - By definition, RTS needs to happen all the time
- Table list affects both synchronous and asynchronous collection
- No defined maintenance window effectively disables asynchronous collection
  - Results in only synchronous collection
  - Inaccurate statistics will persist longer in the statistics cache

Some examples:
The following causes the online maintenance to occur for 3 hours at the end of the first day of every month when ever falls on Monday:

```xml
<OnlineWindow Occurrence="During" startTime="21:00:00" duration="3">
  <DaysOfWeek>Mon</DaysOfWeek>
  <DaysOfMonth>1</DaysOfMonth>
  <MonthsOfYear>All</MonthsOfYear>
</OnlineWindow>
```

You can specify which tables to exclude from the automatic statistics collection by using an expression similar to an SQL-style "where clause" in the FilterCondition. For example, the following specifies that all tables with names that match the pattern 'EMP%' should be excluded from the statistics collection:

```xml
<RunstatsTableScope>
  <FilterCondition>TABSCHEMA NOT LIKE 'EMP%' </FilterCondition>
</RunstatsTableScope>
```

You can specify `<FilterCondition/>` to select all the tables.

For example, the following specifies that statistics should be collected for all tables, including system tables:

```xml
<RunstatsTableScope>
  <FilterCondition/>
</RunstatsTableScope>
```

For example, the following specifies that statistics should be collected for all tables except system tables:

```xml
<RunstatsTableScope>
  <FilterCondition>TABNAME NOT LIKE 'SYS%' </FilterCondition>
</RunstatsTableScope>
```
RTS and Statistical Profiles

- Statistical profiles allow RUNSTATS options to be registered in a profile
- Profile can be specified for subsequent RUNSTATS without re-specifying options
  - Profile can be specified for LOAD
- Profile is stored in system catalogs
  - SYSIBM.SYSTABLES.STATISTICS_PROFILE
  - Stored in form of original RUNSTATS command
- Profile can be modified or replaced
- All RUNSTATS options can be specified
  - Including sampling and index options

The RUNSTATS utility provides a statistical profile facility to:
1) register a statistical profile, while optionally gathering statistics
2) modify an existing statistical profile stored in the catalogs, while optionally gathering statistics
3) repeatedly gather statistics on the table using an already registered statistics profile for that particular table.

This may be convenient for multiple scripts that need to perform RUNSTATS on the same set of tables so the RUNSTATS options don’t need to be repeated in every script. Additionally, the statistical profile can be specified for LOAD so that consistent RUNSTATS options can be used and don’t need to be repeated on the LOAD command.

When a statistical profile is registered, a RUNSTATS command string corresponding to that profile is at the same time built and stored in the STATISTICS_PROFILE column of the catalog table SYSIBM.SYSTABLES. An internal version of the profile is also maintained in the system catalogs in SYSTABLES.PACKED_DESC.
Statistical Profile Examples

- Register a profile and gather statistics:
  RUNSTATS ON TABLE db2user.employee
  WITH DISTRIBUTION ON COLUMNS (EMPL_TITLE, EMPL_SALARY)
  DEFAULT NUM_FREQVALUES 50 SET PROFILE
- Use the registered profile to gather statistics:
  RUNSTATS ON TABLE db2user.employee USE PROFILE
- Update the profile without gathering statistics:
  RUNSTATS ON TABLE db2user.employee
  ON COLUMNS (EMPL_NAME LIKE STATISTICS, (EMPL_TITLE, EMPL_SALARY))
  WITH DISTRIBUTION ON COLUMNS (EMPL_TITLE NUM_FREQVALUES 75)
  UPDATE PROFILE ONLY
- Unset a profile (new to V9.5)
  RUNSTATS ON TABLE db2user.employee UNSET PROFILE

Statistics Options:
---+--------- -----------+---+---------------------+-----><
'-| (Other Options) |-'   '-| Profile Options |-'

Profile Options:

|-- SET PROFILE NONE ---------------------.
 |                          |
|--+--------------------------------------|
+-+-| SET |------ PROFILE ++--+-++---+
 |   |             |             |
 '---| UPDATE |-'    '---| ONLY |-'
RTS and Statistical Profiles

• RTS and automatics statistics collection use statistical profiles, when available
  • Default RUNSTATS options used otherwise
  • RUNSTATS ON TABLE db2user.employee WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL
• Exceptions for RTS
  • Synchronous statistics collection may use a different sampling rate and method than what is specified in the statistical profile
    • Necessary to limit overhead
  • Statistics fabrication may not be possible for all columns specified in the statistical profile
    • E.g. column must be leading in an index in order to fabricate COLCARD, HIGH2KEY and LOW2KEY
• Asynchronous collection always follows the options specified in the statistical profile

Synchronous and asynchronous statistics are collected according to a statistical profile that is in effect for a table, with the following exceptions:

To minimize the overhead of synchronous statistics collection, the database manager might collect statistics using sampling. In this case, the sampling rate and method might be different from those specified in the statistical profile.

Synchronous statistics collection might choose to fabricate statistics, but it might not be possible to fabricate all statistics specified in the statistical profile. For example, column statistics such as COLCARD, HIGH2KEY, and LOW2KEY cannot be fabricated unless the column is leading in some index.

If synchronous statistics collection cannot collect all statistics specified in the statistical profile, an asynchronous collection request is submitted.
System and Event Monitor Changes

- **Database level**
  - `stats_cache_size` – Size of statistics cache
  - `stats_fabrications` – Total number of statistics fabrications
  - `sync_runstats` – Total number of synchronous RUNSTATS operations
  - `async_runstats` – Total number of asynchronous RUNSTATS operations

- **Database and statement level**
  - `stats_fabricate_time` – Total time spent on statistics fabrication activities
  - `sync_runstats_time` – Total time spent on synchronous RUNSTATS activities

Use these elements along with `total_exec_time` and `num_executions` to evaluate the impact of synchronous or fabricated RUNSTATS on query performance.


**stats_cache_size** – Size of statistics cache monitor element
The current size of the statistics cache, which is used in a catalog partition to cache statistics information generated by real-time statistics gathering.

**stats_fabrications** – Total number of statistics fabrications monitor elements
The total number of statistics fabrications performed by real-time statistics during query compilation for all the database applications. Rather than obtaining statistics by scanning data stored in a table or an index, statistics are fabricated based on metadata maintained by the index and data manager. Values reported by all the database partitions are aggregated together.

**sync_runstats** – Total number of synchronous RUNSTATS activities monitor element
The total number of synchronous RUNSTATS activities triggered by real-time statistics gathering for all the applications in the database. This value includes both successful and unsuccessful synchronous RUNSTATS commands. Values reported by all the database partitions are aggregated together.

**async_runstats** – Total number of asynchronous RUNSTATS requests monitor element
The total number of successful asynchronous RUNSTATS activities performed by real-time statistics gathering for all the applications in the database. Values reported by all the database partitions are aggregated together.

**stats_fabricate_time** – Total time spent on statistics fabrication activities monitor element
The total time spent on statistics fabrications by real-time statistics gathering, in milliseconds. Statistics fabrication is the statistics collection activity needed to generate statistics during query compilation. If this monitor element is collected at the database level, it represents the total time spent on real-time statistics gathering activities for all the applications running on the database. If it is collected at the statement level, it represents the time spent on the latest real-time statistics gathering activities for the statement. The times reported by all the database partitions are aggregated together.

**sync_runstats_time** – Total time spent on synchronous RUNSTATS activities monitor element
The total time spent on synchronous RUNSTATS activities triggered by real-time statistics gathering, in milliseconds. The synchronous RUNSTATS activities occur during query compilation. At the database level, this monitor element represents the total time spent on synchronous RUNSTATS activities for all the applications running on the database, triggered by real-time statistics gathering. At the statement level, it represents the time spent on the latest synchronous RUNSTATS activities for a particular statement, triggered by real-time statistics gathering. Values reported by all the database partitions are aggregated together.
ADMIN_GET_TAB_INFO Table Function

• ADMIN_GET_TAB_INFO_V95 table function and ADMINTABINFO administrative view
• New STATSTYPE column
  • Indicates the type of statistics collected for a table
  • ‘F’ = Fabricated statistics
  • ‘A’ = Asynchronously gathered statistics.
  • ‘S’ = Synchronously gathered statistics.
  • ‘U’ = User gathered statistics.
  • RUNSTATS, CREATE INDEX, LOAD, REDISTRIBUTE or by manually updating system catalog statistics.

db2pd Extensions

- **db2pd –tcbstats**
  - New RTSUDI counter
    - The number of rows updated, deleted or inserted since the last RTS collection, asynchronous collection or manual RUNSTATS
  - Existing UDI counter
    - The number of rows updated, deleted or inserted since the last asynchronous collection or manual RUNSTATS

- **db2pd -statisticscache summary | detail | find**
  - Displays contents of statistics cache

  schema=<schema> object=<object>
Explain Facility Enhancements

- Capture all statistics used for query optimization
  - Could be sourced from the statistics cache
  - Statistics cache version will be different from the system catalogs
- All statistics are stored in the explain snapshot
- Collect explain snapshot in addition to explain table population
- Methods:
  - `SET CURRENT EXPLAIN MODE EXPLAIN`
  - `SET CURRENT EXPLAIN SNAPSHOT EXPLAIN <query>`
  - Or:
    - `EXPLAIN PLAN WITH SNAPSHOT FOR <query>`
- `db2exfmt` or Visual Explain will display the statistics
- `EXPLAIN_FORMAT_STATS` scalar function
  - Can be used to format snapshot directly from explain tables

---


**EXPLAIN_FORMAT_STATS** Scalar function

This new scalar function is used to display formatted statistics information which is parsed and extracted from explain snapshot captured for a given query. The data type of the result is `CLOB(50M)`.

**Syntax**

```sql
EXPLAIN_FORMAT_STATS(snapshots)

The schema is SYSPROC.
```

**Scaler function parameters**

*snapshot*

An input argument of type `BLOB(10M)` that is the explain snapshot captured for a given query.

It is stored as `snapshot` column of explain table `EXPLAIN_STATEMENT`

**Authorization**

EXECUTE privilege on the `EXPLAIN_FORMAT_STATS` function.

**Example**

```sql
SELECT EXPLAIN_FORMAT_STATS(SNAPSHOT) FROM EXPLAIN_STATEMENT WHERE EXPLAIN_REQUESTER = 'DB2USER1' AND EXPLAIN_TIME = timestamp('2006-05-12-14.38.11.109432') AND SOURCE_NAME = 'SQLC2F0A' AND SOURCE_SCHEMA = 'NULLID' AND SOURCE_VERSION = '' AND EXPLAIN_LEVEL = 'O' AND STMTNO = 1 AND SECTNO = 201
```
Explain and RTS Activity

• RTS can occur during explain processing
  • If the query is being explained and run
  • REOPT option

<table>
<thead>
<tr>
<th>CURRENT EXPLAIN MODE</th>
<th>RTS Considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>EXPLAIN</td>
<td>NO</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>REOPT</td>
<td>YES</td>
</tr>
<tr>
<td>RECOMMEND INDEXES</td>
<td>NO</td>
</tr>
<tr>
<td>EVALUATE INDEXES</td>
<td>NO</td>
</tr>
</tbody>
</table>
Statistics Logging Facility

- Logs all statistics collection activities
  - Synchronous, asynchronous or manual RUNSTATS
- High speed log – no contention from multiple agents
- Rotating log
- Default name is db2optstats.<number>.log
- Resides in $DIAGPATH/events directory
  - Typically sqllib/db2dump/events
- Log behavior is controlled by DB2_OPTSTATS_LOG registry variable
  - Specify number of log files, size of log file, name and location

`db2set
DB2_OPTSTATS_LOG=ON,NUM=6,SIZE=25,NAME=mystatslog,
DIR=mystats`


**DB2_OPTSTATS_LOG**

Operating system: All

Default=Not set (see details below), Values=OFF, ON {NUM | SIZE | NAME | DIR}

**DB2_OPTSTATS_LOG** specifies the attributes of the statistics event logging files which are used to monitor and analyze statistics collection related activities. When **DB2_OPTSTATS_LOG** is not set or set to ON, statistics event logging is enabled, allowing you to monitor system performance and keep a history for better problem determination. Log records are written to the first log file until it is full. Subsequent records are written to the next available log file. If the maximum number of files is reached, the oldest log file will be overwritten with the new records. If system resource consumption is of great concern to you, disable this registry variable by setting it to OFF.

When statistics event logging is explicitly enabled (set to ON), there are a number of options you can modify:

- **NUM**: the maximum number of rotating log files. Default=5, Values 1 - 15
- **SIZE**: the maximum size of rotating log files. (The size of each rotating file is SIZE/NUM.) Default=100 Mb, Values 1 Mb – 4096 Mb
- **NAME**: the base name for rotating log files. Default=db2optstats.<number>.log, for instance db2optstats.0.log, db2optstats.1.log, etc.
- **DIR**: the base directory for rotating log files. Default=$DIAGPATH/events

You can specify a value for as many of these options as you want, just ensure that ON is the first value when you want to enable statistics logging. For instance, to enable statistics logging with maximum of 6 log files, a maximum file size of 25 Mb, a base file name of mystatslog, and the directory mystats, issue the following command:

`db2set DB2_OPTSTATS_LOG=ON,NUM=6,SIZE=25,NAME=mystatslog,DIR=mystats`
Statistics Logging Facility

- Statistics log can be viewed directly or
- Statistics log records can be retrieved with a table function
  - SYSPROC.PD_GET_DIAG_HIST
- Generic table function used for multiple logging facilities

<table>
<thead>
<tr>
<th>EVENTTYPE</th>
<th>OBJTYPE</th>
<th>OBJSCHEMA</th>
<th>OBJNAME</th>
<th>EVENT1</th>
<th>EVENT2_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>STATS DAEMON</td>
<td>-</td>
<td>PROD_DB</td>
<td>2007-07-09-18.37.40.398905 start</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE AND INDEX STATS</td>
<td>DB2USER</td>
<td>DISTRICT</td>
<td>2007-07-09-18.37.43.261222 Synchronous start</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE AND INDEX STATS</td>
<td>DB2USER</td>
<td>DISTRICT</td>
<td>2007-07-09-18.37.43.407447 Synchronous success</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE AND INDEX STATS</td>
<td>DB2USER</td>
<td>CUSTOMER</td>
<td>2007-07-09-18.37.43.471614 Asynchronous start</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE AND INDEX STATS</td>
<td>DB2USER</td>
<td>CUSTOMER</td>
<td>2007-07-09-18.37.43.524496 Asynchronous success</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>STATS DAEMON</td>
<td>-</td>
<td>PROD_DB</td>
<td>2007-07-09-18.37.43.526212 success</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE STATS</td>
<td>DB2USER</td>
<td>ORDER_LINE</td>
<td>2007-07-09-18.37.48.676524 Synchronous sampled start</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE STATS</td>
<td>DB2USER</td>
<td>ORDER_LINE</td>
<td>2007-07-09-18.37.53.677546 Synchronous sampled failure</td>
<td></td>
</tr>
<tr>
<td>START</td>
<td>EVALUATION</td>
<td>-</td>
<td>PROD_DB</td>
<td>2007-10-12-23.36.11.092739 success</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE AND INDEX STATS</td>
<td>DB2USER</td>
<td>DISTRICT</td>
<td>2007-10-12-23.36.30.736003 Asynchronous start</td>
<td></td>
</tr>
<tr>
<td>COLLECT</td>
<td>TABLE AND INDEX STATS</td>
<td>DB2USER</td>
<td>DISTRICT</td>
<td>2007-10-12-23.36.34.029556 Asynchronous success</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>EVALUATION</td>
<td>-</td>
<td>PROD_DB</td>
<td>2007-10-12-23.36.39.681988 success</td>
<td></td>
</tr>
</tbody>
</table>


In this example, the query returns statistics log records for events up to one year prior to the current timestamp, by invoking PD_GET_DIAG_HIST.

```
SELECT pid, tid, substr(eventtype, 1,10), 
substr(objtype,1,30) as objtype, 
substr(objname_qualifier,1,20) as objschema, 
substr(objname,1,10) as objname, 
substr(first_eventqualifier,1,26) as event1, 
substr(second_eventqualifiertype,1,2) as event2_type, 
substr(second_eventqualifier,1,20) event2, 
substr(third_eventqualifiertype,1,6) event3_type, 
substr(third_eventqualifier,1,15) event3, 
substr(eventstate,1,20) as eventstate FROM 
TABLE( SYSPROC.PD_GET_DIAG_HIST ( 'optstats', 'EX', 'NONE', CURRENT_TIMESTAMP - 1 year, CAST( NULL AS TIMESTAMP ))) as sl order by 
timestamp(varchar(substr(first_eventqualifier,1,26), 26)) ;
```
DPF Considerations

- Only 1 DB agent performs synchronous statistics collection or fabrication per table
- Synchronous or fabricated statistics are collected from a single DB partition
  - Existing RUNSTATS limitation
- A consistent DB partition is chosen for all synchronous, asynchronous or fabrication requests, regardless of DB partition where sensitivity analysis occurred
  - ‘Statistics reference DB partition’
  - Ensures consistent statistics across RTS actions
  - First partition in DB partition group
- Sensitivity analysis uses UDI and RTSUDI from statistics reference DB partition
Exceptions

- No synchronous collection support (including fabrication) for nicknames and statistical views
- No asynchronous collection support for statistical views
  - New in V9.5 - asynchronous support for nicknames
  - Asynchronous support for statistical views in future release
- No asynchronous collection support for DGTTs (pre-V9.5 limitation)
- RTS activities begin 5 mins after DB activation
- REORG (offline or online) doesn’t trigger RTS or automatic statistics collection
  - Data statistics may not have changed
  - Long term direction is to have REORG collect statistics
- Automatic REORG triggers automatics statistics collection
Considerations

- Do explicit DB activations to ensure statistics cache isn't freed if DB implicitly deactivated
Summary

• Real-time statistics enhances existing automatic statistics collection functionality
• RTS provides statistics quickly with low overhead
• New features to access statistics cache
• New features to monitor and track all statistics collection activities
Session C03
Real-time Statistics Collection

John Hornibrook
IBM Canada
jhornibr@ca.ibm.com