Control Query Default (CQD) Reference Guide

Version 2.0.0

Table of Contents

License Statement

Version	Date
1.3.0	January, 2016

Chapter 1. About This Document

This guide describes Trafodion Control Query Defaults (CQDs) that are used to override system-level default settings.

1.1. Intended Audience

This guide is intended for database administrators and application programmers who want to use CQDs to override system-default settings when querying a Trafodion database.

1.2. New and Changed Information

This is a new guide.

1.3. Notation Conventions

This list summarizes the notation conventions for syntax presentation in this manual.

- UPPERCASE LETTERS

  Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required.

  ```
  SELECT
  ```

- lowercase letters

  Lowercase letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required.

  ```
  file-name
  ```

- [ ] Brackets

  Brackets enclose optional syntax items.

  ```
  DATETIME [start-field TO] end-field
  ```

  A group of items enclosed in brackets is a list from which you can choose one item or none.

The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines.

For example:

```
DROP SCHEMA schema [CASCADE]
DROP SCHEMA schema [ CASCADE | RESTRICT ]
```

- { } Braces

  Braces enclose required syntax items.

  ```
  FROM { grantee [, grantee ] ... }
  ```

  A group of items enclosed in braces is a list from which you are required to choose one item.

  The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines.

  For example:

  ```
  INTERVAL { start-field TO end-field }
  { single-field }
  INTERVAL { start-field TO end-field | single-field }
  ```

- | Vertical Line

  A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces.

  ```
  {expression | NULL}
  ```

- … Ellipsis

  An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times.

  ```
  ATTRIBUTE[S] attribute [, attribute] ...
  {, sql-expression } ...
  ```

  An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of

times.

For example:

```
expression-n ...
```

- Punctuation

  Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown.

```
DAY (datetime-expression)
@script-file
```

  Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown.

  For example:

```
"{" module-name [, module-name] ... "}"
```

- Item Spacing

  Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma.

```
DAY (datetime-expression) DAY(datetime-expression)
```

  If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
myfile.sh
```

- Line Spacing

  If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line.

  This spacing distinguishes items in a continuation line from items in a vertical list of selections.

```
match-value [NOT] LIKE _pattern
   [ESCAPE esc-char-expression]
```

1.4. Publishing History

Product Version	Publication Date
Trafodion Release 1.3.0	January 2016.

1.5. Comments Encouraged

The Trafodion community encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to:

issues@trafodion.incubator.apache.org

Include the document title and any comment, error found, or suggestion for improvement you have concerning this document. Or, even better, join our community and help us improve our documentation. Please refer to Trafodion Contributor Guide for details.

Chapter 2. Introduction

Refer to the Trafodion SQL Reference Manual full documentation of the CQD (CONTROL QUERY DEFAULT) statement.

The CONTROL QUERY DEFAULT statement changes the default settings for the current process. You can execute the CONTROL QUERY DEFAULT statement in a client-based tool like TrafCI or through any ODBC or JDBC application.

Syntax

```
{ CONTROL QUERY DEFAULT | CQD } control-default-option

control-default-option is:
  attribute {'attr-value' | RESET}
```

The result of the execution of a CONTROL QUERY DEFAULT statement stays in effect until the current process terminates or until the execution of another statement for the same attribute overrides it. CQDs are applied at compile time, so CQDs do not affect any statements that are already prepared. For example:

```
PREPARE x FROM SELECT * FROM t;
CONTROL QUERY DEFAULT SCHEMA 'myschema';
EXECUTE x;                              -- uses the default schema SEABASE
SELECT * FROM t2;                       -- uses MYSCHEMA;
PREPARE y FROM SELECT * FROM t3;
CONTROL QUERY DEFAULT SCHEMA 'seabase';
EXECUTE y;                              -- uses MYSCHEMA;
```

Examples

- Change the maximum supported length of the column names to 200 for the current process:

```
CONTROL QUERY DEFAULT HBASE_MAX_COLUMN_NAME_LENGTH '200';
```

- Reset the HBASE_MAX_COLUMN_NAME_LENGTH attribute to its initial value in the current process:

```
CONTROL QUERY DEFAULT HBASE_MAX_COLUMN_NAME_LENGTH RESET;
```

2.1. CQD Descriptions

The following information is provided for each CQD:

Description	Describes the purpose of the CQD.
Values	Identifies this information: • Values, in the form of a character string, that specify the applicable attribute values for the CQD. • The default attribute value. • If applicable, the Trafodion release in which the attribute values or default changed.
Usage	Describes the conditions when the CQD is helpful, and how to detect the conditions.
Production Usage	Identifies when the CQD is not safe to be used as a permanent setting in production.
Impact	Describes any positive and negative implications of using the CQD.
Level	Indicates one of these levels at which the CQD should be used: • Query • Session • Service • Any NOTE: This level indicates that the CQD can be used at the Query, Session or Service level as long as you fully understands the scope of the impact of the CQD.
Conflicts/Synergies	Describes CQDs that are in conflict with or can be used in conjunction with the CQD.
Real Problem Addressed	Describes any design or solution that the CQD may be a workaround for and how you can directly address the real problem.
Introduced In Release	Indicates the Trafodion release when the CQD was introduced.
Deprecated In Release	Indicates in what release the CQD was deprecated.

Chapter 3. Query Plans

This section describes CQDs that are used to influence query plans.

3.1. DEFAULT_DEGREE_OF_PARALLELISM

Description	Defines the minimum size of the adaptive segment; that is, the number of processors available for query operator parallelism. The optimizer may choose an adaptive-segment size that is equal to, or the multiple of, the value of this CQD depending on the maximum estimated resource consumed by any single operator in the query. The optimizer may also decide to run the query with no parallelism if the resource consumption estimate is very low.
Values	Unsigned Integer. The default value is **'16'**.
Usage	For systems running at higher levels of concurrency with workloads that include a large number of small queries, reducing the default degree of parallelism may help achieve higher throughput. With the default of 16, for 32-node systems, adaptive segmentation can use two 16-node virtual segments to execute queries that do not require a degree of parallelism of 32. This default setting can, for example, be changed to 8 for a 16-node system, to allow adaptive segmentation to leverage a lower degree of parallelism.
Production Usage	Not applicable.
Impact	Lowering the value of this CQD can increase the throughput of high-concurrency small-query workloads, but has the potential disadvantage of increasing the elapsed time for some of the longer running queries that leverage adaptive segmentation.
Level	System. There may be scenarios where you want to influence the degree of adaptive segmentation parallelism only for a certain set of queries and use it at the service level.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.2. HASH_JOINS

Description	Determines whether the Trafodion Optimizer considers Hash Join when generating an execution plan.
Values	**'ON'** Hash Join is considered. **'OFF'** Hash Join is not considered. The default value is **'ON'**.
Usage	Use this CQD when you want to force the optimizer to generate a query plan that does not use any Hash Joins.
Production Usage	Hash Join is an important join implementation strategy for most complex queries. It is highly recommended that you do not turn HASH_JOINS OFF; that is, this CQD should be used to force a query plan for a particular query on an exception basis only.
Impact	Turning HASH_JOINS OFF may result in very inefficient query plans that use expensive nested joins or sorts for merge joins.
Level	Query.
Conflicts/Synergies	If you turn all three join implementations OFF (Hash Joins, Nested Joins, and Merge Joins), then the compiler may fail to generate query plans.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.3. HBASE_COPROCESSORS

Description	Allow HBase coprocessors to be used when computing aggregates.
Values	**'ON'**: Use HBase coprocessors. **'OFF'**: Do not use HBase coprocessors. The default value is **'ON'**.
Usage	Enables Trafodion to use HBase coprocessors to do early aggregation and filtering at the HBase Region Server level. This CQD does not affect Transaction coprocessors used by Trafodion. As of Release 2.0 only COUNT(*) queries will be affected by this attribute.
Production Usage	Yes.
Impact	Network traffic between Region Server and Trafodion processes is reduced but the Region Server can become very busy when aggregating over large tables.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.4. HIVE_NUM_ESPS_PER_DATANODE

Description	Determines number of ESP processes used to scan a Hive table, per HDFS DataNode.
Values	Positive integer. The default value is **'2'**.
Usage	Use this CQD to increase or decrease the number of scanners that process a single Hive table. If a Hive scan is found to be the bottleneck for a particular query, then increasing this attribute to; for example, 4 or higher will help. On the other hand decreasing the attribute to 1 could help with concurrency.
Production Usage	Yes.
Impact	Controls number of ESPs and, therefore, affects query execution time and system workload.
Level	Query.
Conflicts/Synergies	The CQD HIVE_MIN_BYTES_PER_ESP_PARTITION (default = 67108864) may need to be adjusted downward when this attribute is used to increase the parallelism of scanning smaller Hive tables.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.5. JOIN_ORDER_BY_USER

Description	Enables or disables the join order in which the optimizer joins the tables to be the sequence of the tables in the FROM clause of the query.
Values	**'ON'** Join order is forced. **'OFF'** Join order is decided by the optimizer. The default value is **'OFF'**.
Usage	When set to ON, the optimizer considers only execution plans that have the join order matching the sequence of the tables in the FROM clause.
Production Usage	This setting is to be used for forcing a desired join order that was not generated by default by the optimizer only. It can be used as a workaround for query plans with inefficient join order.
Impact	Because you are in effect forcing the optimizer to use a plan that joins the table in the order specified in the FROM clause, the plan generated may not be the optimal one.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.6. MC_SKEW_SENSITIVITY_THRESHOLD

Description	Define the multi-column skew sensitivity threshold T used by multi-column skew-insensitive hash join (Skew Buster). Let `f` be the occurrence frequency of a skew value `v`, `DoP` be the degree of parallelism of a hash join operator, and `RC` be the row count of the source data (for example, fact table) where the skew originates. The hash join will run in the anti-skew mode for `v` if `f >= T * DoP / RC`.
Values	**< 0**: Disable the multi-column skew buster. **>= 0**: Define the threshold T. Default value: **0.1**.
Usage	Use of a **negative value** to disable multi-column anti-skew hash joins. This may slow down query performance when multi-column skew values are present in the fact table. A value of **0** treats every multi-column value as skew values. This may increase network traffic since skewed values are broadcasted from the inner side child of the hash join to all join processes. A value **greater than 0** selects those multi-column values as skewed values if their occurrence frequencies are high enough.
Production Usage	Consult the Trafodion community.
Impact	This CQD impacts runtime performance.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.7. MDAM_NO_STATS_POSITIONS_THRESHOLD

Description	This CQD effects the behavior of the query optimizer when there are no statistics available for a query having range predicates on key columns. The Trafodion Optimizer calculates the worst case number of seeks that the MDAM access method would do if chosen for the query. If this number is greater than the value of MDAM_NO_STATS_POSITIONS_THRESHOLD, then MDAM is not considered for the query execution plan.
Values	Any integer greater than equal to zero. Default value: **10**.
Usage	In certain situations, queries on tables lacking statistics may not be optimal because MDAM was not chosen. Increasing the value for this CQD allows MDAM to be chosen in more cases. On the other hand, if the value is made too high and the worst case scenario actually occurs, an MDAM plan may perform poorly.
Production Usage	Consult the Trafodion community.
Impact	Table scans on tables lacking statistics may improve by varying the value of this CQD. Results vary depending on the actual data in the table and the semantics of the query.
Level	Query.
Conflicts/Synergies	If MDAM_SCAN_METHOD is set to **'OFF'**, then this CQD has no effect.
Real Problem Addressed	Perform UPDATE STATISTICS on the table (at the very least on key columns) to obtain statistics.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.8. MDAM_SCAN_METHOD

Description	Enables or disables the Multi-Dimensional Access Method (MDAM).
Values	**'ON'** MDAM is considered. **'OFF'** MDAM is disabled. The default value is **'ON'**.
Usage	In certain situations, the Trafodion Optimizer might choose MDAM inappropriately, causing poor performance. In such situations you may want to turn MDAM OFF for the query it is effecting.
Production Usage	Yes.
Impact	Table scans with predicates on non-leading clustering key column(s) could benefit from MDAM access method if the leading column(s) has a small number of distinct values. Turning MDAM off results in a longer scan time for such queries.
Level	Set this CQD at the query level when MDAM is not working efficiently for a specific query. However, there may be cases (usually a defect) where a larger set of queries is being negatively impacted by MDAM. In those cases you may want to set it at the service or system level.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.9. MERGE_JOINS

Description	Determines if Merge Join is considered by the optimizer to generate an execution plan.
Values	**'ON'**: Merge Join is considered. **'OFF'**: Merge Join is disabled. The default value is **'ON'**.
Usage	Use this CQD when you want to force a query plan not to use Merge Joins. This is useful as a workaround for query plans with very expensive sorts for Merge Joins. Turning MERGE_JOINS OFF also has the advantage of reducing the query compile time.
Production Usage	Merge Join is an efficient join implementation strategy if the physical schema was designed to take advantage of it. For example, large tables are physically ordered based on the most frequently joined column(s).
Impact	Turning MERGE_JOINS OFF may result in the optimizer not considering potentially efficient query plans, for queries with large joins on tables that are physically ordered by the join column(s). Turning MERGE_JOINS ON causes an increase in compile time because the optimizer now has to consider many more join options.
Level	Set this CQD at the query level when a Merge Join is not working efficiently for a specific query. However, there may be cases (usually a defect) where a larger set of queries is being negatively impacted by Merge Joins. In those cases you may want to set it at the service or system level.
Conflicts/Synergies	Avoid turning all the three join implementations OFF (Hash Joins, Nested Joins, and Merge Joins). This may result in the Trafodion Compiler failing to generate query plans.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.10. NESTED_JOINS

Description	Determines if Nested Join is considered by the optimizer to generate an execution plan.
Values	**'ON'**: Nested Join is considered. **'OFF'**: Nested Join is disabled. The default value is **'ON'**.
Usage	Use this CQD when you want to force a query plan not to use Nested Joins. This is useful as a workaround for query plans with very expensive Nested Joins, which may occur if the optimizer fails to estimate the cost of a Nested Join correctly.
Production Usage	Nested Join is an important join implementation strategy for many complex queries. It is recommended not to turn NESTED_JOINS OFF. It should only be used to force a query plan for a particular query on an exception basis.
Impact	Turning NESTED_JOINS OFF may result in inefficient query plans for certain type of queries, such as light workloads and star join queries.
Level	Query.
Conflicts/Synergies	Avoid turning all the three join implementations OFF (Hash Joins, Nested Joins, and Merge Joins). This may result in the compiler failing to generate query plans.
Real Problem Addressed	The problem of inefficient Nested Joins can be better handled using a higher degree of query plan robustness as set by the ROBUST_QUERY_OPTIMIZATION CQD.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.11. OPTIMIZATION_LEVEL

Description	Controls the optimizer resources and time spent for optimizing a query plan, with level 0 indicating the least amount of optimization effort and level 5 indicating the most. Lower optimization levels produce lower plan quality with minimal compile time, while higher optimization levels cause the compiler to spend more compilation time to produce better plan quality.
Values	'0', '2', '3', '5' The default value is **'3'**.
Usage	Reduce the optimization level when compile time is longer than desired and queries have relatively small execution cost and are simple in structure.
Production Usage	Use this CQD only as a workaround for queries with unacceptable compile time or plan quality.
Impact	Lowering the optimization level below the system default may result in inefficient query execution plans. Increasing the optimization level over the system default may result in very high compile time for complex queries.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.12. PARALLEL_NUM_ESPS

Description	Controls the maximum number of parallel ESPs that work on a particular operation; for example, a join.
Values	Unsigned Integer: The maximum number of ESPs that should be used for a particular operation. The value must be less than the number of nodes in the cluster. **'SYSTEM'**: The compiler calculates the number of ESPs to be used. The default value is **'SYSTEM'**.
Usage	Used to control the maximum degree of parallelism for a query. This could be useful to limit the number of resources (node and memory) any single query can use.
Production Usage	Not applicable.
Impact	Lowering the value of this CQD can increase the throughput of high concurrency small and medium query workloads, but has the potential disadvantage of increasing the elapsed time of some of the long-running queries.
Level	Service.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.13. RISK_PREMIUM_NJ

Description	Influences the optimizer to choose other types of joins over nested joins, by making a nested join plan relatively more expensive.
Values	Any positive fractional value. The default value is **'1.2'**.
Usage	Review ROBUST_QUERY_OPTIMIZATION before considering the use of this CQD. The default setting indicates that a nested join plan must be 20% cheaper before it is allowed to win over competing safer (hash) join plans. A setting of 1.0 means no handicap for nested joins. A setting of 5.0 means a nested join must be 400% cheaper before it is allowed to win over competing hash join plans. If it is determined that the optimizer is using nested joins often enough where these plans are resulting in poor performance, then this CQD may be used to influence the optimizer to consider another join instead, such as a hash join, in some of those cases. NESTED_JOINS OFF could turn nested joins off completely. However, there are many cases where nested joins do provide better performance than hash joins, and turning them off completely may negatively impact the performance of queries that can do a lot better with nested joins.
Production Usage	Consult with the Trafodion community.
Impact	Specifying a risk premium insures against nested joins being chosen when they should not have been. However, this can also result in nested joins not being chosen where the cardinality estimation was in fact accurate and a nested join could have performed better. Therefore, this setting should be used with care in order to get robustness with a net gain in performance.
Level	Any. There may be cases where there are different applications or workloads that might benefit from this CQD more than other workloads. In such cases this could be used at the Service level.
Conflicts/Synergies	ROBUST_QUERY_OPTIMIZATION is a CQD that provides a robust query setting across the board, influencing the nested join risk premium as well. It is advisable that you use that setting instead to influence plans, unless they are specifically addressing nested join issues and need to use this setting independent of that CQD.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.14. RISK_PREMIUM_SERIAL

Description	Influences the optimizer to choose a parallel plan over a serial plan, by making a serial plan relatively more expensive.
Values	Any positive fractional value. The default value is **'1.2'**.
Usage	Review ROBUST_QUERY_OPTIMIZATION before considering the use of this CQD. The default setting means that a serial plan must be 20% cheaper before it is allowed to win over competing parallel plans. A setting of 1.0 means no handicap for serial plans. A setting of 2.0 means a serial plan must be 100% cheaper before it is allowed to win over competing parallel plans. If it is determined that the optimizer is using serial plans often enough where these plans are resulting in poor performance, then this CQD may be used to influence the optimizer to consider parallel plans instead in some of those cases.
Production Usage	Consult with the Trafodion community.
Impact	Specifying a risk premium insures against serial plans being chosen when they should not have been. However, this can also result in serial plans not being chosen where the cardinality estimation was in fact accurate and a serial plan could have performed better. Therefore, this setting should be used with care in order to get robustness with a net gain in performance.
Level	Any. There may be cases where there are different applications or workloads that might benefit from this CQD more than other workloads. In such cases this could be used at the Service level.
Conflicts/Synergies	ROBUST_QUERY_OPTIMIZATION is a CQD that provides a robust query setting across the board, influencing the serial plan risk premium as well. It is advised that you use that setting instead to influence plans, unless they are specifically addressing serial plan issues and need to use this setting independent of that CQD.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.15. RISK_PREMIUM_SERIAL_SCALEBACK_MAXCARD_THRESHOLD

Description	Defines the minimal estimated max cardinality or row count of any relational operators in a query above which the risk premium for serial plan is applied. A serial query plan is favored by the Trafodion Compiler when it estimates the query reads and processes small amount of data. The estimation error could become large when some operator is calculated to produce many rows yielding a non-optimal serial plan. This CQD helps prevent utilizing serial plan in such cases.
Values	An unsigned integer value.
Usage	Adjust this CQD when necessary only. Use of a value smaller than the default (10,000) to penalize more serial plans or favor more parallel plans for operators produce less number of rows. Otherwise, use of a larger value.
Production Usage	Consult with the Trafodion community.
Impact	This CQD impacts plan quality.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.16. ROBUST_QUERY_OPTIMIZATION

Description	Provides a simpler way to influence the optimizer's choice of query plans. The optimizer chooses query plans based on cardinality estimates (the number of result rows estimated at each step of a query execution plan). Actual cardinalities encountered at query execution often differ from estimates. The optimizer considers risky choices and exacts risk premiums before it chooses a plan that is inherently sensitive to cardinality estimation errors. Risky choices include: **Nested joins**: Can be excellent data reducers but they can also result in extremely long-running queries when their outer table cardinality is grossly underestimated. **Serial plans**: Are an excellent choice because they use the least resources when processing low data volumes. But they can also result in very long-running queries when actual cardinalities greatly exceed estimates. **Complete sharing of ESP partitioning schemes**: Correct parallel processing requires partitioning the data across ESP instances using a partitioning scheme usually based on the largest table's partitioning keys, join columns, and group by columns. Complete sharing of ESP partitioning schemes minimizes the overhead of runtime repartitioning. But, it can also result in very long-running queries if the "least common denominator" partitioning scheme results in a few active ESPs doing most of the work. This can happen when repartitioning is being done on a very low unique entry count attribute. For example, gender. ROBUST_QUERY_OPTIMIZATION can be used to influence the premiums associated with these risky plans and thereby overall plan quality and performance for your specific workloads.
Values	**'MIN'**: No risk premium. **'HIGH'** and **'MAXIMUM'**: Higher risk premium. **'SYSTEM'**: Safe risk premium. The default value is **'SYSTEM'**.

Usage	MAXIMUM tells the optimizer to make the safest choice of query plans. This means: • RISK_PREMIUM_NJ is set to 5.0: nested join must be 400% cheaper before it can win over competing (hash) join plans. • RISK_PREMIUM_SERIAL is set to 2.0: serial plan must be 100% cheaper before it can win over competing parallel plans. • PARTITIONING_SCHEME_SHARING is set to 2: no partition scheme sharing between adjacent ESP fragments. HIGH tells the optimizer to make a safer choice of query plans. This means: • RISK_PREMIUM_NJ is set to 2.5: nested join must be 150% cheaper before it can win over competing (hash) join plans. • RISK_PREMIUM_SERIAL is set to 1.5: serial plan must be 50% cheaper before it can win over completing parallel plans. • PARTITIONING_SCHEME_SHARING is set to 1: subset sharing of partition schemes between adjacent ESP fragments. SYSTEM tells the optimizer to make a safe choice of query plans. This means: • RISK_PREMIUM_NJ is set to 1.2: nested join must be 20% cheaper before it can win over competing (hash) join plans. • RISK_PREMIUM_SERIAL is set to 1.2: serial plan must be 20% cheaper before it can win over completing parallel plans. • PARTITIONING_SCHEME_SHARING is set to 1: subset sharing of partition schemes between adjacent ESP fragments. MIN tells the optimizer to believe its cardinality estimates are always correct when choosing query plans. For example, don’t apply any risk premium for risky operations. This means: • RISK_PREMIUM_NJ is set to 1.0: nested join can win over competing (hash) join plans purely based on cost & cardinality estimates. • RISK_PREMIUM_SERIAL is set to 1.0: serial plan can win over completing parallel plans purely based on cost & cardinality estimates. • PARTITIONING_SCHEME_SHARING is set to 0: complete sharing of partition schemes between adjacent ESP fragments. If histograms are accurate and the queries are relatively simple, then you could choose a lower robustness setting. In complex query environments where queries could end up processing large amounts of data, you should consider higher settings. If you notice that when queries are not performing well it is due to either nested join plans, serial plans, or reduced parallelism, then you could consider increasing risk premiums to see if you can get overall better performance.
Production Usage	It is best to try out different options to achieve best overall performance in a test environment before implementing the changed settings in a production environment.
Impact	Specifying a risk premium insures against nested joins or serial plans being chosen when they should not have been. However, this can also result in such plans not being chosen where the cardinality estimation was in fact accurate and such plans could have performed better. So this setting should be used with care in order to get robustness with a net gain in performance.
Level	Any. There may be cases where there are different applications or workloads that might benefit from this CQD more than other workloads. In such cases this could be used at the Service level.

Conflicts/Synergies	This conflicts with the RISK_PREMIUM_NJ, RISK_PREMIUM_SERIAL, and PARTITIONING_SCHEME_SHARING settings. Use this CQD when possible. Use the risk premium settings rarely, when specific premiums need to be set differently to address specific issues. If overall this CQD is working well but you have outliers, such as poor nested join plans or inappropriate serial plans, then you could use the individual CQDs at a finer granularity, such as at a query level, to get better plans.
Real Problem Addressed	Sometimes the cardinality underestimation, compared to the actual row counts, resulting in a nested join or serial plan being chosen when it shouldn't have been, may be due to not enough, or inaccurate, histogram statistics information available to the optimizer. So, first and foremost, histogram statistics should be kept up to date along with the multi-column statistics that the optimizer may warn about. However, cardinality underestimations may still happen at higher levels of an execution plan.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.17. SKEW_EXPLAIN

Description	Turns on the reporting of anti-skew join plan details in EXPLAIN or EXPLAIN OPTIONS 'f'.
Values	**'OFF'**: Disables the use of SKEW_EXPLAIN. **'ON'**: Enables skew information in EXPLAIN. The default value is **'OFF'**.
Usage	Not applicable.
Production Usage	Not applicable.
Impact	Query plans are not changed by this CQD.
Level	Any.
Conflicts/Synergies	Allows additional information to be displayed in explain plans. It has no impact on query plans.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.18. SKEW_ROWCOUNT_THRESHOLD

Description	The optimizer looks for skewed values and address that skew if the number of rows in the table exceeds this threshold.
Values	**'n'**: where n is the number of rows Default: **'1000000'**
Usage	Skew can occur either in a nested join or a hash join. Currently, such skew is typically handled for the outer table of a join. The outer table is identified by the Explain plan. The first indication is that there is a performance problem caused by skew that is not addressed by the compiler. A skew can be detected by observing the imbalanced use of node cycles during query execution or by observed table skew. The default setting has been chosen to handle most skew values that are worth worrying about. That is, in other cases there may be skew but the impact on total query execution may be minimal. However, there could be cases where this is not true. If you detect or suspect that a performance issue is caused by skew issues, then you need to look at the cardinality of the table with skew (typically, the outer table in a join) to determine whether the table has fewer rows than defined by this CQD (default: 1 million rows.) If the table has fewer rows than the CQD setting, then set this CQD to a value smaller than the number of rows in that table. If changing the setting addresses the performance problem, then the skew has been addressed. You can also examine whether the optimizer has addressed the skew issue by turning on SKEW_EXPLAIN EXPLAIN of the plan and then run the EXPLAIN statement. If you find that you need to change the setting of this CQD to a value other than the default value, then please file a case providing information about the table skew, the query, and the value that worked. Filing a case for this situation helps us tune the default value further.
Production Usage	See usage discussion above. Use this CQD with care.
Impact	A lower setting allows more skews to be detected and addressed. However, compile time is increased.
Level	Any.
Conflicts/Synergies	SKEW_EXPLAIN can be used as described in above. Also, SKEW_SENSITIVITY_THRESHOLD is relevant if this threshold allows a skew to be detected only.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.19. SKEW_SENSITIVITY_THRESHOLD

Description	Used to specify a threshold that determines whether a value in the join column is considered skewed.
Values	**'-1'**: Disables the use of skew buster. **'n'**: 'n' is a floating-point value greater than or equal to 0. The default value is **'-1'**.
Usage	A value is considered skewed if its occurrence frequency is greater than the threshold value multiplied by the average number of rows per processing node (CPU). That is, if the average row count equals the row count divided by the number of processing nodes, then the frequency of the value is greater than the threshold multiplied by the average row count. A setting of *n*, where *n* >= 0, indicates that the value should be considered as skewed if its occurrence frequency is greater than *n* times the average number of rows per processing node. If some small skew is suspected for hash joins during query execution (detected by observing spiked CPU busy usage), then try to lower this setting. A default setting of 0.1 should eliminate most skews. Setting the CQD to a very large value, such as 10, is not recommended, as it effectively turns off skew buster.
Production Usage	Not applicable.
Impact	Not applicable.
Level	Any.
Conflicts/Synergies	This CQD is only relevant if the SKEW_ROWCOUNT_THRESHOLD has been met. The SKEW_ROWCOUNT_THRESHOLD CQD controls the row count of the table at which the optimizer looks for a skew.
Real Problem Addressed	Skew is quite common in a real big-data application, and is effectively addressed by skew buster. However, there may be design opportunities that could help address the problem as well.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.20. SUBQUERY_UNNESTING

Description	Allows correlated subqueries in a SQL statement to be unnested, so that they can be executed efficiently.
Values	**'ON'**: Unnesting enabled. **'OFF'**: Unnesting disabled. The default value is **'ON'**.
Usage	Turn this CQD OFF when (in rare cases) unnesting a correlated subquery causes performance to degrade. If this attribute has to be turned OFF, then that could indicate a bug in the Trafodion Optimizer. Consult with the Trafodion community.
Production Usage	Yes.
Impact	Turn OFF with caution at a system level, as other queries which rely on un-nesting could be adversely impacted.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.21. TRAF_ALLOW_ESP_COLOCATION

Description	Enables or disables whether ESP are colocated with HBase Region Servers, which minimizes the inter-node network traffic between the ESP processes and the HBase Region Servers.
Values	**'ON'**: Colocation enabled. **'OFF'**: Colocation disabled. The default value is **'OFF'**.
Usage	Enable the feature when each region server serves approximately equal amount of data, and/or reducing network traffic is important.
Production Usage	Consult with the Trafodion community.
Impact	Plan quality.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.22. TRAF_UPSERT_WITH_INSERT_DEFAULT_SEMANTICS

Description	Enables population of omitted columns in an UPSERT statement with default values when the table is created in aligned-row format mode.
Values	**'ON'**: Default population enabled. **'OFF'**: Default population disabled. The default value is **'ON'**.
Usage	When a column with default value is omitted in an UPSERT statement of a table in aligned row format mode, then the statement is transformed to merge. If the row already exists, then the omitted columns are populated with values from the existing row. If the row doesn’t exist, then the omitted columns are populated with default values. This default behavior can be changed by setting this CQD to 'ON', which improves the performance of the UPSERT statements with omitted default value columns.
Production Usage	Yes.
Impact	Improved upsert performance of aligned row format tables.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

3.23. UPD_ORDERED

Description	Controls whether rows should be inserted, updated, or deleted in clustering key order.
Values	**'ON'**: The optimizer generates and considers plans where the rows are inserted, updated, or deleted in clustering key order. **'OFF'**: The optimizer does not generate plans where the rows must be inserted, updated, or deleted in clustering key order. The default value is **'ON'**.
Usage	Inserting, updating or deleting rows in the clustering key order is most efficient and highly recommended. Turning this CQD OFF may result in saving the data sorting cost but at the expense of having less efficient random I/O Insert/Update/Delete operations. If you know that he data is already sorted in clustering key order, or is mostly in clustering key order, so that it would not result in random I/O, you could set this CQD to OFF.
Production Usage	Not applicable.
Impact	If turned OFF, the system may perform large number of inefficient Random I/Os when performing Insert/Update/Delete operations.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 4. Query Execution

This section describes CQDs that are used to influence query execution.

4.1. HBASE_ASYNC_OPERATION

Description	Allows index maintenance to be performed concurrently with base table operation.
Values	**'ON'** Index maintenance is allowed. **'OFF'** Index maintenance is not allowed. The default value is **'ON'**.
Usage	HBase `put` operations are blocking. When the table has one or more indexes, then the insert/update/delete (IUD) operation response time is improved by executing the index maintenance operations concurrently with the base table operation: the put operations to these HBase tables are executed in different threads.
Production Usage	Yes. It is 'ON' by default. This feature can be disabled by setting this CQD to 'OFF'.
Impact	IUD operations on tables with one or more indexes can become slower.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

4.2. HBASE_CACHE_BLOCKS

Description	Influences HBase to retain the data blocks in memory after they are read.
Values	**'ON'/'OFF'/'SYSTEM'** The default value is **'SYSTEM'**.
Usage	HBase maintains the block cache structure to retain the data blocks in memory after they are read. In LRU block cache configuration, the amount of block cache retained in memory is proportional to the amount of reserved maximum Java heap size of the region server. LRU Block Cache is the default in HBase. The Trafodion Optimizer determines whether a sequential scan of the HBase table in a query would cause the full eviction of the data blocks cached earlier thereby impacting the performance of the random reads. The cache blocks option is turned off for the table in such a case. Set the CQD HBASE_REGION_SERVER_MAX_HEAP_SIZE value to reflect the amount of java heap size reserved for the region servers. This CQD is used by the Trafodion Optimizer to evaluate if the block cache should be turned off.
Production Usage	Leave the setting to be 'SYSTEM' when HBase is configured to use LRU block cache. If needed, you can override this settings with 'ON' or 'OFF'. With other HBase configurations, you need to set HBASE_CACHE_BLOCKS to 'ON' or 'OFF' based on your application needs.
Impact	Automatically retains the random read performance.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

4.3. HBASE_FILTER_PREDS

Description	Allows push down of predicates to HBase Region Servers using HBase filters and optimize the columns retrieved from Region Servers. Only supported for NON ALIGN FORMAT tables.
Values	**'OFF'**: Predicates are never pushed down. **'ON'**: A first implementation targeted for deprecation is enabled. Support simple predicate formed by a combination of AND only. Could be counter-productive when applied on nullable columns. **'1'**: Same as **'ON'**. **'2'**: Full feature is enabled. An explain plan can show whether predicates are successfully pushed down to the Region Servers and what columns are really retrieved. The default value is **'OFF'**.
Usage	Used to improve performance by reducing the number of columns retrieved to a strict minimum and filter out rows as early as possible.
Production Usage	Please consult the Trafodion community.
Impact	Using this CQD increases the amount of work done in the HBase Region Servers.
Level	System or Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 2.0.0.
Deprecated In Release	Not applicable.

4.4. HBASE_HASH2_PARTITIONING

Description	Treat salted Trafodion tables as hash-partitioned on the salt columns.
Values	**'OFF'**: Salted Trafodion tables are not hash-partitioned on the salt columns. **'ON'**: Salted Trafodion tables are hash-partitioned on the salt columns. The default value is **'ON'**.
Usage	If, for any reason, there are issues with parallel plans on salted tables (especially with data skew) then try setting this CQD to OFF.
Production Usage	Yes.
Impact	Not applicable.
Level	System or Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 2.0.0.
Deprecated In Release	Not applicable.

4.5. HBASE_NUM_CACHE_ROWS_MAX

Description	Determines the number of rows obtained from HBase in one RPC call to the HBase Region Server in a sequential scan operation,
Values	Numeric value. The default value is **'10000'**.
Usage	This CQD can be used to tune the query to perform optimally by reducing the number of interactions to the HBase Region Servers during a sequential scan of a table. You need to consider how soon the maximum number of rows are materialized on the Region Servers. When filtering is pushed down to Region Servers, then it can take a longer time depending upon the query and the predicates involved. This can result in HBase scanner timeouts.
Production Usage	Use the default setting and reduce the value to avoid HBase scanner timeouts. Consult with the Trafodion community if you think that you need to use this CQD.
Impact	Not applicable.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

4.6. HBASE_ROWSET_VSBB_OPT

Description	Allows INSERT, UPDATE, and DELETE (IUD) operations to be performed as an HBase batch `put` operation.
Values	**'ON'**: Perform IUD operations as an HBase batch `put` operation. **'OFF'**: Do not perform IUD operations as an HBase batch `put` operation. The default value is **'ON'**.
Usage	When IUD operation involves multiple tuples, then the Trafodion Optimizer evaluates whether these operations can be done in a batch manner at the HBase level thereby reducing the network interactions between the client applications and the HBase Region Servers. If possible, then the query plan involves VSBB operators. The Virtual Sequential Block Buffer(VSBB) name is retained in Trafodion though it is unrelated to HBase.
Production Usage	Yes.
Impact	IUD operations can become slower if this CQD is set to 'OFF'.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

4.7. HBASE_ROWSET_VSBB_SIZE

Description	Determines the maximum number of rows in a batch `put` operation to HBase.
Values	Numeric value. The default value is **'1024'**.
Usage	The Trafodion execution engine already adjusts the number of rows in a batch depending upon how fast the queue to IUD (INSERT,UPDATE,DELETE) operator is filled up in the data flow architecture of Trafodion. You can adjust the maximum size to suit your application needs and thus tune it to perform optimally.
Production Usage	Yes. You can disable this feature by setting the HBASE_ROWSET_VSBB_OPT CQD to 'OFF'.
Impact	The performance of your application may be affected by setting this CQD too low.
Level	Query.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

4.8. HBASE_SMALL_SCANNER

Description	Enables Trafodion to leverage the HBase small scanner optimization. This optimization reduces I/O usage up to 66% and enables non-blocking reads for higher concurrency support. When a scan is known to require less than a HBASE BLOCK SIZE (default is 64K), then enabling the HBase small scanner optimization increases performance.
Values	**'OFF'**: Never use the HBase small scanner optimization. **'SYSTEM'**: Only enable the HBase small scanner optimization when the Trafodion Compiler determines that the scan size will fit in the table's HBASE BLOCK SIZE **'ON'**: Enable the HBase small scanner optimization regardless of the size of scan. The default value is **'OFF'**.
Usage	Consider using this CQD to improve the performance of your queries.
Production Usage	Consult with the Trafodion community.
Impact	The performance of small scan may increase by 1.4x. This CQD can be very useful for MDAM scans.
Level	System or Session.
Conflicts/Synergies	MDAM performance may be improved by 1.4x when correctly picking HBase block size so that each MDAM scan operation fit within a HBASE BLOCK SIZE boundary. If you enable small scanner on large size scan incorrectly, then you are likely to see a 6% performance decrease. The returned results will still be correct.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 2.0.0.
Deprecated In Release	Not applicable.

Chapter 5. Manage Histograms

This section describes CQDs that are used to manage histograms.

5.1. CACHE_HISTOGRAMS_REFRESH_INTERVAL

Description	Defines the time interval after which timestamps for cached histograms are checked for refresh processing.
Values	Unsigned integer in seconds. The default value is **'3600'** (1 hour).
Usage	Histogram statistics are cached so that the compiler can avoid access to the metadata tables, thereby reducing compile times. The timestamp of the tables are checked against those of the cached histograms at an interval specified by this CQD, in order to see if the cached histograms need to be refreshed. You can increase the interval to reduce the impact on compile times as long as you do not need to obtain fresh statistics more frequently in order to improve query performance. It may be that the default interval is too long and you would rather refresh the statistics more frequently than the default one hour, in order to improve query performance at the cost of increased compile times. This setting depends on how frequently you are updating statistics on tables. There is no point in refreshing statistics frequently when statistics are not being updated during that time. On the other hand, if you are updating statistics, or generating them for the first time on freshly loaded tables frequently enough, and you want these to be picked up immediately by the compiler because you have seen this to have a dramatic impact on plan quality, then you can make the refresh more frequent.
Production Usage	Not applicable.
Impact	Longer histogram refresh intervals can improve compile times. However, longer refresh intervals yield more obsolete the histograms. More obsolete histograms may result in poor performance for queries that could leverage recently updated statistics.
Level	System or Service.
Conflicts/Synergies	Frequency of update statistics run using MAINTAIN.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

5.2. HIST_MISSING_STATS_WARNING_LEVEL

Description	Controls the level of missing statistics warnings that should be displayed. The warnings impacted are 6007, 6008, 6010, and 6011.
Values	**'0'**: Display no warnings. **'1'**: Display only missing single column statistics warnings. These include 6008 and 6011. **'2'**: Display all missing single and multi-column statistics warnings for scans only. **'3'**: Display all missing single and multi-column statistics warnings for scans and join operators only. **'4'**: Display all missing single and multi-column statistics warnings. The default value is **'4'**.
Usage	If you do not want to see these warnings, then change the setting to '0'. If you want to track the warnings, then you have a choice of which warnings you want to track. Each setting gives you the ability to filter the warnings seen for missing single or multi-column statistics for join or scan operations. This controls the resulting number of warning messages. If poor query plans are being caused by cardinality estimations that seem to be off, then you can check the histogram statistics to see if statistics are being collected for those columns and how accurate they are. If you don’t find statistics being collected, then you could look for the warnings by setting this CQD to the appropriate setting. Based on that you could take appropriate action: either find out why USAS is not collecting appropriate statistics, or if USAS is not being used then ensure that update statistics is being run to generate those statistics.
Production Usage	Many tools divide a query into several steps. During the first phases volatile tables are created and populated, the last phase usually joins all the volatile tables created in the previous steps. Usually statistics are not needed for those volatile tables because the final join is straight forward and the optimizer has no big choices. Nevertheless the log is flooded with useless warnings if you don’t set the warning level to 0. If possible, try to direct queries from those tools to a dedicated service where you set the warning level to 0.
Impact	Though the warnings give information about all statistics that are missing, it can be overwhelming to get several warnings. Not all warnings may contribute to plan improvements. The optimizer issues multi-column statistics warnings based on the search path, some of which may not even impact the plan quality. Also, the cost of gathering statistics on those columns may not bring commensurate benefit to a large number of queries.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

5.3. HIST_NO_STATS_REFRESH_INTERVAL

Description	Defines the time interval after which the fake histograms in the cache should be refreshed unconditionally.
Values	Unsigned integer. Unit is seconds. The default value is **'3600'** (1 hour).
Usage	Histogram statistics are "fake" when update statistics is not being run, but instead the customer is updating the histogram tables directly with statistics to guide the optimizer. This may be done if the data in the table is very volatile (such as for temporary tables), update statistics is not possible because of constant flush and fill of the table occurring, and statistics are manually set to provide some guidance to the optimizer to generate a good plan. If these fake statistics are updated constantly to reflect the data churn, then this default can be set to 0. This would ensure that the histograms with fake statistics are not cached and are always refreshed. If these fake statistics are set and not touched again, then this interval could be set very high.
Production Usage	Not applicable.
Impact	Setting a high interval improves compilation time. However, if statistics are being updated, then the compiler may be working with obsolete histogram statistics, potentially resulting in poorer plans.
Level	Service.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

5.4. HIST_PREFETCH

Description	Influences the compiler to pre-fetch the histograms and save them in cache.
Values	**'ON'**: Pre-fetches the histograms. **'OFF'**: Does not pre-fetch the histograms. The default value is **'ON'**.
Usage	You may want to turn this off if you don’t want to pre-fetch a large number of histograms, many of which may not be used.
Production Usage	Not applicable.
Impact	Though it makes compilation time faster, it may result in the histogram cache to be filled with histograms that may never be used.
Level	System or Service.
Conflicts/Synergies	Use this CQD with CACHE_HISTOGRAMS. If CACHE_HISTOGRAMS is OFF, then this CQD has no effect.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

5.5. HIST_ROWCOUNT_REQUIRING_STATS

Description	Specifies the minimum row count for which the Trafodion Optimizer needs histograms, in order to compute better cardinality estimates. The Optimizer does not issue any missing statistics warnings for tables whose size is smaller than the value of this CQD.
Values	Integer. The default value is **'50000'**.
Usage	Use this CQD to reduce the number of statistics warnings.
Production Usage	Not applicable.
Impact	Missing statistics warnings are not displayed for smaller tables, which in most cases don't impact plan quality much. However, there may be some exceptions where missing statistics on small tables could result in less than optimal plans.
Level	System.
Conflicts/Synergies	Use this CQD with HIST_MISSING_STATS_WARNING_LEVEL. If the warning level CQD is 0, then this CQD does not have any effect. Also, for tables having fewer rows than set in this CQD, no warnings are displayed irrespective of the warning level.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

5.6. HIST_USE_SAMPLE_FOR_CARDINALITY_ESTIMATION

Description	Enables the Compile Time Stats feature. Compile Time Stats are produced during query plan generation by executing a subset of the query on a subset of data to gather more accurate cardinality estimations.
Values	**'ON'**: Compile Time Statistics is enabled. **'OFF'**: Compile Time Statistics is disabled. The default value is **'ON'**.
Usage	The feature is very helpful for cases when the query contains complex predicates on a table. These predicates include LIKE, CASE, any other expressions or more than one range predicates and equality on large character columns. It can be disabled if most of the queries are single table or at most two-way joins. It can also be disabled if the extra collection of statistics seems to be adversely affecting the total query compile and execution time.
Production Usage	Not applicable.
Impact	The feature improves cardinality estimates for Scan operators thus improving the plan quality. However, it can also increase the compile time.
Level	Any.
Conflicts/Synergies	In order to use the feature in its default form, sample tables should exist in `public_access_schema`.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 6. Transaction Control and Locking

This section describes CQDs that are used for transaction control and locking.

6.1. ISOLATION_LEVEL

Description	Specifies the default transaction isolation level that queries use.
Values	**'READ UNCOMMITTED'**, **'READ COMMITTED'**, **'REPEATABLE READ'**, or **'SERIALIZABLE'**. The default value is **'READ COMMITTED'** (ANSI).
Usage	If you use uncommitted access (reading "dirty" data when queries are accessing data that is being simultaneously updated), then you can set the default isolation level as READ UNCOMMITTED. The default isolation level of READ COMMITTED can cause concurrency issues because reads would wait on locked rows. If rows are locked by long-running transactions with infrequent commits, this can cause severe concurrency issues for SELECT queries. See Conflicts/Synergies.
Production Usage	Not applicable.
Impact	Using this CQD has implications on locking and concurrency. If set to READ UNCOMMITTED, then select queries read through locks and don't have to wait on locks. But they won't see committed consistent data. If set to READ COMMITTED (the default setting), then the reads wait on locked rows before they proceed with the scan. The read can proceed only when the rows locked by another transaction are released after that transaction commits. The reader does not lock rows. If set to REPEATABLE READ or SERIALIZABLE, then it has severe implications on concurrency because every row read is also locked.
Level	While you can use this at a query or a service level, the most common use is a system-wide setting. If query tools are being used, then the query level setting cannot be used. A service level setting may provide uncommitted access to certain users while providing the default committed access to the other users, depending which users need to see consistent data. If however, access to tables during updates is well controlled and read uncommitted is acceptable, then this can be set at the system level.
Conflicts/Synergies	The problem with using READ UNCOMMITTED as the isolation level default value is that in a SET TRANSACTION statement, the only possible access mode is READ ONLY. Any query that attempts to update the database would fail. To facilitate updates and DDL statements while the isolation level is set to READ UNCOMMITTED, a new default attribute ISOLATION_LEVEL_FOR_UPDATES is provided. This default attribute specifies the isolation level for update and DDL statements. If not specified, or if not present in the SYSTEM_DEFAULTS table, the default value is the same as the ISOLATION_LEVEL default attribute. However, if specified or present in the SYSTEM_DEFAULTS table, then its value is used as the isolation level for updates and DDL statements. UPDATE in ISOLATION_LEVEL_FOR_UPDATES refers to INSERT, UPDATE, and DELETE statements.
Real Problem Addressed	Not applicable.

Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

6.2. ISOLATION_LEVEL_FOR_UPDATES

Description	Specifies the default transaction isolation level for these update operations: INSERT, UPDATE, or DELETE.
Values	**'READ UNCOMMITTED'**, **'READ COMMITTED'**, **'REPEATABLE READ'**, or **'SERIALIZABLE'**. The default value is **'READ COMMITTED'** (ANSI).
Usage	Set this CQD to READ UNCOMMITTED to prevent users from performing any updates.
Production Usage	Not applicable.
Impact	When set, this CQD prevents users from doing any of these update operations: INSERT, UPDATE, or DELETE.
Level	Service.
Conflicts/Synergies	Works with the ISOLATION_LEVEL setting. Both settings are READ COMMITTED by default. ISOLATION_LEVEL can be set to READ UNCOMMITTED. This CQD still remains READ COMMITTED. You can change it to READ UNCOMMITTED to prevent queries running at the service level to not perform any updates.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 7. Runtime Controls

This section describes CQDs that are used for runtime controls.

7.1. LAST0_MODE

Description	Ensures that all parts of the query plan are executed but no rows are returned by the query.
Values	**'ON'** or **'OFF'**. The default value is **'OFF'**.
Usage	This setting provides a realistic measure of the query's performance, minus the cost/time of returning the rows to the client. It is especially useful for testing the plans and performance of queries that return large result sets.
Production Usage	Only use this CQD to assess the performance of a query.
Impact	The query runs completely but no rows are returned.
Level	Query.
Conflicts/Synergies	Not to be confused with SELECT [LAST 0] which behaves the same way but does not guarantee that the plan is the same as when you do not use the [LAST 0] clause in the query.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

7.2. QUERY_LIMIT_SQL_PROCESS_CPU

Description	Use to limit the amount of CPU time that a query is allowed to use in any one server process (MXESP) including the NDCS server (MXOSRVR). Also includes the CPU time the query spends in the disk process (ESAM). If a query exceeds the limit, then an error is raised and the query is terminated. This is a way to limit the impact on the system of a poorly written or badly optimized query.
Values	**'0'**: There is no limit. **Greater than *'0'** and up through **'2,147,483,583'**: The limit, in seconds, to how much CPU time a query is allowed. The default value is **'0'**.
Usage	This setting helps you with queries that are poorly written or are badly optimized. A poorly written query does not use predicates to limit the number of rows processed. A query that joins large tables without a predicate can have a severe impact on the system. A badly optimized query can result from failure to maintain histograms. Typically, these are ad-hoc queries. These types of queries seldom run to completion, and are instead stopped after the problems that they cause to other users of the system are noticed.
Production Usage	Not applicable.
Impact	Use of this default can prevent any one query from using an unlimited amount of CPU time. However, if the default is set too low, then even well behaved, useful queries fail to complete.
Level	Service.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Use standard processed for creating high-quality queries, and procedures that ensure that table histograms are always current.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 8. Schema Controls

This section describes CQDs that are used for schema controls.

8.1. Catalog

Description	Specifies the default catalog name for all DDL and DML statements.
Values	Any valid ANSI name, including delimited names. The default is **'TRAFODION'**.
Usage	Trafodion tables must be in a catalog called TRAFODION. If you mostly access Hive or native HBase tables, then the catalog could be changed to HIVE or HBASE respectively. The default setting is overridden by any catalog name specified in a SQL statement.
Production Usage	Yes.
Impact	Not applicable.
Level	Any.
Conflicts/Synergies	Alternately you can use the SET CATALOG statement.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

8.2. Schema

Description	Sets the default schema for the session.
Values	SQL identifier. The default is **'TRAFODION'**.
Usage	A SET SCHEMA statement or a CONTROL QUERY DEFAULT SCHEMA statement can be used to override the default schema name.
Production Usage	It is a convenience so you do not have to type in two-part names.
Impact	Not applicable.
Level	Any.
Conflicts/Synergies	Alternately you can use the SET SCHEMA statement.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 9. Table Definition

This section describes CQDs that are used for table definition

9.1. ALLOW_NULLABLE_UNIQUE_KEY_CONSTRAINT

Description	Allow Trafodion tables to be created with NULLABLE columns in the PRIMARY or STORE BY key.
Values	**'ON'** or **'OFF'** The default value is **'OFF'**.
Usage	Allows NULLABLE columns to be included in the PRIMARY or STORE BY key for Trafodion tables. This CQD must be set prior to creating the table. It is not necessary to specify this CQD during DML operations on the table. When performing UPDATE STATISTICS with SAMPLE option on such tables, this CQD must be set so that an appropriate sample table can be created.
Production Usage	Please consult with the Trafodion community.
Impact	Two bytes are added to the key for each nullable column.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

9.2. HBASE_BLOCK_SIZE

Description	Allow Trafodion tables to be created with specified HBase block size.
Values	Positive integer. The default value is **'65536'**.
Usage	The value of this attribute is passed on to HBase when a Trafodion table is created in HBase. See the Apache HBase™ Reference Guide for usage information .
Production Usage	Yes.
Impact	The impact depends on the type of table access. Choose a block size that is appropriate for how the table is primarily accessed.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

9.3. HIVE_DEFAULT_CHARSET

Description	Defines what character set the columns of Hive tables should have.
Values	**'ISO88591'** or **'UTF8'** Default: **'UTF8'**
Usage	Set this to ISO88591 when reading from Hive tables with ISO8859-1 data.
Production Usage	Yes.
Impact	Not applicable.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

9.4. HIVE_FILE_CHARSET

Description	For certain character sets that are not supported in Trafodion, you can specify the character set here causing Trafodion to automatically convert the data to the character set specified in the HIVE_DEFAULT_CHARSET CQD. This feature is currently supported only for **'GBK'**, and only if HIVE_DEFAULT_CHARSET is set to **'UTF8'**.
Values	empty or **'GBK'** Default: empty
Usage	Leave this blank, unless you want to access GBK data in Hive tables.
Production Usage	Yes.
Impact	Not applicable.
Level	System.
Conflicts/Synergies	Only applicable if HIVE_DEFAULT_CHARSET is set to **'UTF8'**.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

9.5. HIVE_MAX_STRING_LENGTH

Description	Hive columns of type STRING have a maximum length in Trafodion, which you can specify with this CQD. **NOTE** For UTF-8 data, this length is specified in bytes, not UTF-8 characters.
Values	1-n. Default: 32000.
Usage	Set this to the lowest possible value to improve system performance.
Production Usage	Yes.
Impact	Not applicable.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 10. Update Statistics and Reorg

10.1. USTAT_MAX_READ_AGE_IN_MIN

Description	When performing update statistics with the NECESSARY keyword or with automation, this is the number of minutes that are allowed to have elapsed since a histogram was marked as read for it to be regenerated. Histograms that were marked more than USTAT_MAX_READ_AGE_IN_MIN minutes ago are not regenerated.
Values	0 through max unsigned integer. Setting this CQD to a value less than 2 * USTAT_AUTOMATION_INTERVAL is silently ignored and 2 * USTAT_AUTOMATION_INTERVAL (default value of 2880) is used. The default value is 5760 (4 days).
Usage	Influences how frequently the histograms for a table are regenerated. If a table is being used frequently, then chances are that its histograms are also be considered for update frequently. However, if a table is not used frequently, then this CQD influences how frequently the histograms for that table are updated. A smaller setting reduces the number of histograms being updated if there are many tables that have not been used within that interval. A larger setting updates histogram for many more tables that are not being accessed that often.
Production Usage	Consult the Trafodion community.
Impact	Influences the number of histograms that need to be regenerated and therefore the time it takes for update statistics automation to regenerate histograms for all the tables that so qualify.
Level	System.
Conflicts/Synergies	USTAT_AUTOMATION_INTERVAL sets a lower bound on this CQD. It is influenced by USTAT_AUTO_READTIME_UPDATE_INTERVAL, which influences how often READ_TIME is updated for the histogram. This CQD qualifies the histogram to be regenerated; it is the maximum time since READ_TIME was updated. $$$
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

10.2. USTAT_MIN_ROWCOUNT_FOR_SAMPLE

Description	Sets the minimum rows that need to be in a table before sampling is used to update statistics for that table. If a table has a fewer rows than the value of this CQD, then the SAMPLE option is silently ignored when performing update statistics.
Values	1 through max unsigned integer. The default value is 10000.
Usage	Influences for what tables sampling is used for histogram statistics generation. If the setting is smaller, then more tables qualify for sampling. If the setting is larger, then fewer tables qualify for sampling. Sampling can result in faster update statistics run times. But for smaller tables, it can also result in poor histogram accuracy which could result in poor query plans.
Production Usage	Consult the Trafodion community.
Impact	Setting this CQD to a smaller value means that sampling is used for tables with fewer rows, when the SAMPLE option is specified as part of update statistics. This can result in less accurate histograms and poor query plans, because the sample size may be too small to generate good estimates for histograms. Setting this CQD to a larger value can result in sampling not being used for many tables and therefore longer update statistics run times. However, these tables may also have more accurate histograms.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

10.3. USTAT_MIN_ROWCOUNT_FOR_LOW_SAMPLE

Description	Defines the behavior of the UPDATE STATISTICS utility. It places a lower limit on the number of sample rows that will be used when sampling. If the number of rows in the table is less than this value and sampling is used, then the sample size used will be that specified by the HIST_DEFAULT_SAMPLE_MIN CQD. (which defaults to 10,000).
Values	This CQD may take on any integer value greater than zero. The default for this CQD is **1,000,000**.
Usage	Prevents accidental use of too-small samples when generating statistics on tables. If sample sizes are too small, then histogram statistics will be less accurate, leading to potentially less efficient query plans.
Production Usage	Consult the Trafodion community.
Impact	Setting this CQD to lower values may result in smaller sample sizes for small tables. This may slightly improve UPDATE STATISTICS run time, but at the cost of potentially less efficient queries.
Level	System.
Conflicts/Synergies	HIST_DEFAULT_SAMPLE_MIN effects the behavior of this CQD.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 11. Operational Controls

This section describes CQDs that are used for operational controls.

11.1. AUTO_QUERY_RETRY_WARNINGS

Description	Indicates whether a warning should be issued when a query is retried, in case a failed query is automatically retried.
Values	**'ON'** or **'OFF'** The default value is **'OFF'**.
Usage	There are certain cases, such as when a node failure occurs, where a query may fail midstream. The query is usually retried automatically if it has not returned any data. When such retries happen, then you may want to see a warning that an automatic retry took place. That would be a reason to turn this on.
Production Usage	Not applicable.
Impact	You get a warning message every time a query is automatically retried due to a failure. When there is a node failure, then a large number of queries may be impacted. Therefore, you need to assess if you want to see a flood of warnings. The warning is returned after the query completes.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

11.2. EXPLAIN_DESCRIPTION_COLUMN_SIZE

Description	Specifies maximum length of DESCRIPTION column for EXPLAIN virtual tables.
Values	Positive integer > 10,000 Default: -1, indicating a maximum size of 10,000.
Usage	Query-plan information for a SQL DML statement is stored temporarily in the Explain virtual table. For large queries or queries with complex predicates, the default size of 10 KB may be insufficient to describe certain nodes in the query plan. Specifying a larger value for this CQD allows more bytes to be stored in the description column. Change this setting if you see explain plan being undesirably truncated only.
Production Usage	Yes.
Impact	The explain plan truncation is reduced or removed.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

11.3. HBASE_REGION_SERVER_MAX_HEAP_SIZE

Description	Enables Trafodion to use HBase block cache in an optimal manner. Defines the maximum Java heap size (-Xmx option) the HBase Region Server are assigned, in MB units.
Values	Positive integer. Default: **'1024'**.
Usage	If the HBase Region servers are configured with a maximum Java heap size different than 1 GB, then set this attribute so that Trafodion is aware of the actual maximum heap size of the Region Servers.
Production Usage	Yes.
Impact	When set correctly, this CQD ensures that HBase block cache are be used optimally. Small scans are cached and larger scans are not cached to avoid cache trashing.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

11.4. HIVE_METADATA_REFRESH_INTERVAL

Description	Controls the metadata cache for Hive tables.
Values	**-1**: Never invalidate any cache entries. **0**: Always check the latest meta-data from Hive. **> 0**: The cached Hive meta-data is valid only for <value> seconds. Default: **'0'**.
Usage	Use a value of -1 when the Hive tables are read-only. This allows Trafodion to avoid repeatedly reading the metadata when the table is referenced in multiple queries. Use a value of 0 when updates to the Hive tables are likely to be frequent. Use a positive value *n* to cause Trafodion to re-read metadata after n seconds has elapsed. Use this option when updates to the Hive table happen at least *n* seconds apart.
Production Usage	Consult with the Trafodion community.
Impact	Compilation time.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

11.5. QUERY_CACHE

Description	Attempts to reduce compilation times by storing and reusing previously compiled query plans. It maximizes the chances of plan reuse by parameterizing literals in equality predicates. Two equality predicates, "col = val1" and "col = val2", are considered to match if their selectivities match. A query cache setting of **'16384'** means a maximum of 16,384 KB of compiler memory can be used for keeping previously compiled plans before evicting the oldest unused plan(s) to make room for the latest cacheable plan.
Values	**Up through 4294967295**: Kilobytes of memory allocated to query cache. **'0'**: Turns off query plan caching. The default value is **'16384'** (16 MB).
Usage	To choose the appropriate size for the query cache, examine your applications. Applications that use a PREPARE statement to pre-compile queries once and then EXECUTE the prepared plan, should turn off plan caching. Ad-hoc query applications can specify a size that can hold most of the frequently processed queries. For example, if an application processes 40 classes of queries frequently with an average plan size of 100 KB per query, a cache size of 4000 KB might be optimal. (Plan size is not the same as the size of the SQL statement and is not easy to assess.) There may be applications that are operational in nature, with many small queries, and others that are analytical in nature with large complex queries. Cache size can be set differently for different service levels handling such workloads based on the classes and types of queries, size of the queries, and propensity to get cache hits. Another consideration is how frequently the cache is getting flushed due to the compiler being shutdown and a new one started by an MXOSRVR (ODBC/Connect server), in order to run queries on behalf of a different role than the role that was using the compiler before. If this happens often and not enough static servers can be started to reduce this from happening, then creating a large cache may not be useful, because it has to be flushed and filled too often. After taking the above into account the best way to really assess whether caching is effective, and tune it for your specific applications, is to understand the cache hit statistics, how many queries are forced to be removed from cache (on a least recently used basis), and a number of other statistics about the efficiency of query plan caching for your applications.
Production Usage	Not applicable.
Impact	A larger cache size allows more query plans to be cached. This increases the probability of finding a plan in cache that can be reused for a query, thereby reducing compile time. It does mean that the compiler uses more memory, but because there are usually not that many compilers running in a node, the negative effects may be minimal. However, you do need to know the amount of physical memory available on each node and the number of compilers that run on a node (influenced by the number of concurrent connections configured to run on the cluster). If the cache size is disproportionately large, it is likely to result in reduced performance as the operating system may repeatedly swap the compiler (bloated by a huge cache) in and out of physical memory.
Level	Service.

Conflicts/Synergies	You should be aware that the cache allocated is divided into text caching and template caching. Text caching gets approximately 25% of the cache memory. Query plan caching occurs prior to parsing (text-based caching) and after parsing (template-based caching). The compiler caches same-text queries as text cache hits. Same-text queries are queries whose SQL texts are identical in everything, including case and white space. By caching text-based queries, the compiler avoids redundant re-computation of previously compiled queries and improves performance by reducing compile times and increasing compiler throughput. The text cache is always searched first for a query. If the plan object is not produced due to a text cache miss, then the plan is stored in the template cache if it meets the criteria for template caching.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

11.6. TRAF_LOAD_ALLOW_RISKY_INDEX_MAINTENANCE

Description	Allows incremental index maintenance during bulk load.
Values	**'ON'**: Incremental index maintenance enabled. **'OFF'**: Incremental index maintenance disabled. Default: **'OFF'**.
Usage	When this CQD is ON during a bulk load, then any indexes on a table are maintained incrementally. New rows are added to the base table and all the indexes in HFiles and then during LOAD COMPLETE phase all new files are moved to HBase. Indexes are not offline. However, it causes the index to be inconsistent with the base table if any of the new rows have the same key value as an existing row. Change the default to ON only when certain that new rows do not have a conflict with existing rows in table.
Production Usage	Yes.
Impact	Bulk load into tables with index is faster, when the attribute is set to ON.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

11.7. TRAF_LOAD_FLUSH_SIZE_IN_KB

Description	Specifies the flush size used by bulk load when writing to HFiles.
Values	Positive integer. Default: **'2014'**.
Usage	If the system is not memory constrained, then specifying a larger value may make the write phase of the LOAD statement to proceed faster. Consider using lower values if the table has several indexes and is memory constrained.
Production Usage	Yes.
Impact	Affects memory usage patterns and write performance of LOAD.
Level	System.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

Chapter 12. Debugging

This section describes CQDs that are used for debugging controls.

12.1. UDR_DEBUG_FLAGS

Description	Used when debugging user-defined functions (UDFs).
Values	**0-n** Default: **'0'**.
Usage	See See UDF Tutorial.
Production Usage	Consult the Trafodion community.
Impact	Not applicable.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

12.2. UDR_JVM_DEBUG_PORT

Description	Used when debugging user-defined functions (UDFs).
Values	**0-n** Default: **'0'**.
Usage	See See UDF Tutorial.
Production Usage	Consult the Trafodion community.
Impact	Not applicable.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.

12.3. UDR_JVM_DEBUG_TIMEOUT

Description	Used when debugging user-defined functions (UDFs).
Values	**0-n** Default: **'0'**.
Usage	See See UDF Tutorial.
Production Usage	Consult the Trafodion community.
Impact	Not applicable.
Level	Session.
Conflicts/Synergies	Not applicable.
Real Problem Addressed	Not applicable.
Introduced In Release	Trafodion 1.3.0.
Deprecated In Release	Not applicable.