IBM PureData System for Analytics: Improving Query Performance by Tuning Netezza Optimizer Through Postgres Configuration

The IBM® PureData® System for Analytics, powered by Netezza® technology (“IBM Netezza”), is a data warehouse appliance that features a purpose-built analytics engine. Netezza generates a query plan for all queries that run on the system. The query plan describes the optimal execution query path as determined by the query optimizer component, which relies on statistics that are available about the database objects that are involved in the query.

The Netezza query optimizer is a cost-based optimizer; that is, it determines the cost for the various execution alternatives and chooses the path with the least cost as the optimal execution path for a particular query. The execution plan with the lowest cost among all candidate plans is considered the most efficient. However, in certain situations the Netezza optimizer might not provide the best query plan that leads to the best query performance.

This IBM Redbooks® Analytics Support Web Doc describes some Postgres parameters that can be used to influence the Netezza query optimizer and improve the query performance.

This document is intended for Netezza administrators and application users who run SQL queries on the Netezza database. This document applies to IBM PureData® System for Analytics (powered by Netezza technology) V7.0.3 and higher.

This document covers the following topics:

- Modifying Postgres configuration parameter
- Checking the current settings of a Postgres configuration parameter
- Postgres configuration parameters to make positive influence on a query

Overview

In general, the default optimizer settings are adequate for most operations. However, in some cases you might have information that is unknown to the optimizer or must tune the optimizer for a specific type of statement or workload. Similarly, the optimizer might select different execution plans for some SQL statements when the data volume increases. In such cases, influencing the optimizer might provide better performance if performance problems arise.

You can influence the optimizer by using Postgres configuration parameters at the session level to address single query performance. You also can set the parameters globally by modifying the `postgresql.conf` file.

Figure 1 shows an example of the use of Postgres configuration parameters at the session level, which has precedence over the global settings in the `postgresql.conf` file. In this example, the client uses the ENABLE_FACTREL_PLANNER parameter to influence the optimizer. This parameter is described in "ENABLE_FACTREL_PLANNER".

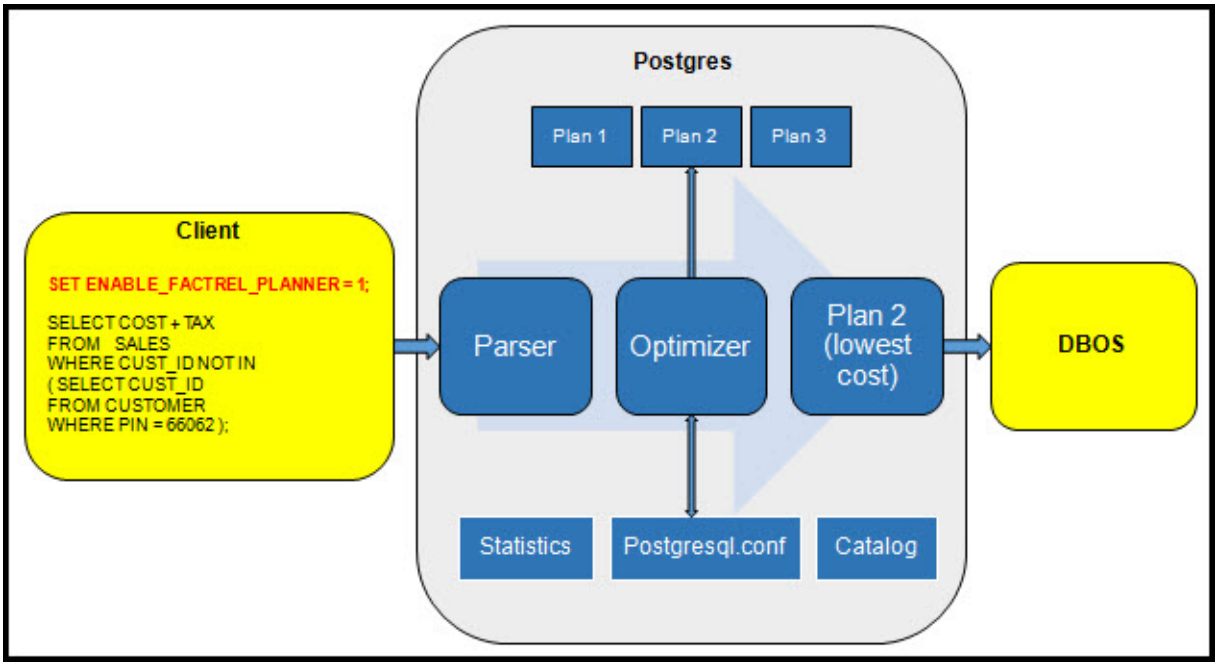

Figure 1 Postgres configuration parameters

Modifying Postgres configuration parameter

A configuration parameter can be modified by using one of the following methods:

- Setting the parameter globally via the configuration file

In this method, the file /nz/data/config/postgresql.conf file is edited. If the parameter you want to modify exists, update it with the appropriate configuration value. If the parameter does not exist in the file, add the parameter at the end. After the /nz/data/config/postgresql.conf file is changed, you can run one of the following commands as user nz to implement the modification:

  - nzstop followed by nzstart (requires a database outage)
  - pkill -HUP postmaster (no outage required)

- Setting parameter at session level

In this method, the configuration parameters are changed to use the SET command in the session level. Run the SET command before the SQL statement is run, as shown in the following example:

```
SET ENABLE_2PHASE_PLANNER = false;
SELECT col1, col2 FROM tab1;
```

Checking the settings of a Postgres configuration parameter

One of the following methods can be used to check the settings of a configuration parameter:

- Use the SHOW command, as shown in the following example:

```
$ nzsql -c "show ENABLE_FACTREL_PLANNER"
NOTICE:  ENABLE_FACTREL_PLANNER is off
```

- Capture the nzdumpschema output, as shown in the following example:

```
$ nzdumpschema db1 | grep -i ENABLE_FACTREL_PLANNER
set ENABLE_FACTREL_PLANNER = 0;
```

- Use the /nz/support/bin/nz_set command, as shown in the following example:

```
$ /nz/support/bin/nz_set | grep -i ENABLE_FACTREL_PLANNER
set ENABLE_FACTREL_PLANNER = off;
```

Postgres configuration parameters to make positive influence on a query

There are many Postgres configuration parameters in Netezza that are configured for different areas; for example, Connection Settings, Security, Memory, Space Map, Checkpoint, and Optimizer. In this section, some common Postgres optimizer configuration parameters that can be use to influence query performance are described.

ENABLE_MERGEJOIN (boolean)

This parameter enables or disables the query planner's use of merge-join plan types. The default setting is ON.

Netezza supports different type of joins; for example HASH, Nested-Loop, and Merge. The optimizer chooses the best algorithm to make all joins in a query. During an equi-join between two tables, a HASH join is used. Most table joins are performed as a HASH join, which is highly optimized for Netezza. However, optimizer might choose merge join over HASH join in the following situations:

- Materialized view
- Data in each join table is sorted
- Distribution key in the tables is defined as floating point data types (“double precision” data types). Floating point numbers have the concept of a +0 and a -0, which makes hashing the values and comparing the values problematical.

A merge join can causes problem when a lot of new data is added to the underlying tables and a materialized view is not refreshed for some time. Then, new data that is sorted during the query execution might take some time and there is a higher cost that is associated with the sort operation that affects the performance.

There are times when the query has many joins. The optimizer might choose a plan that is not optimal and can causes errors that are related to “NOT ENOUGH MEMORY FOR MERGE-STYLE JOIN”.

If this issue occurs, check the planfile to confirm if the optimizer chose a Merge Join and the snippets that are associated with it have a high cost. An excerpt from the `nz_plan` output that confirms that Merge Join was used is shown in the following example:

```
THINGS OF NOTE
==============

Just-In-Time Statistics: 0
Disk Based Join:         0
Cross Product Join:      0
Expression Based Join:   0
Merge Join:              1
MaterializeNode:         0
SUBPLAN_EXPR:            0
Host Distributions:      0


<snip>


3[02]: spu MergeJoinNode
      table 1001679 tblField=0 joinField=0 joinType=0 preSorted=1
      -- (ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY)
      -- Cost=3745839.2..9947649.3 Rows=150.0M Width=0 Size=0 Conf=80
```

Before the query execution, you can disable the Merge Join at the session level to force the optimizer to choose a better plan, as shown in the following example:

```
SET ENABLE_MERGEJOIN = off;
<QUERY GOES HERE> ;
```

ENABLE_FACTREL_PLANNER (boolean)

This parameter enables or disables the Fact Relation (factrel) planner. The factrel planner biases against the data movement of tables with rows that have more than "FACTREL_SIZE_THRESHOLD" settings, which causes the optimizer to favor streaming these tables. Such tables are marked as "FACT" in the plan file. It prevents a broadcast of fact data only. The optimizer ensures that dimension and smaller tables are moved to co-locate the necessary information rather than considering the movement of fact data. The default setting is Off.

Broadcasting fact data is costly in terms of system performance. You can use the factrel to prevent broadcasting fact data. If the ENABLE_FACTREL_PLANNER is set to ON, the optimizer uses the FACTREL_SIZE_THRESHOLD variable (default value of 100 million rows) to establish at what row count a table is considered not suitable for broadcast to other Snippet Processing Units (SPUs) to facilitate the co-locate process on table joins.

After the factrel planner is disabled, the optimizer does not consider the fact table sizes when it generates plans.

If a query was analyzed and a snippet indicates that a particular table is causing a performance issue because of a broadcast, you can adjust the FACTREL_SIZE_THRESHOLD value. This adjustment forces the optimizer to generate a different plan. You can apply this change to an individual session or the entire system. When dealing with tables with a row count close to billion rows, it is better to have factrel planner enabled to avoid possible fact data broadcasts or redistributions. The following example shows how to avoid large table broadcasts:

```
SET ENABLE_FACTREL_PLANNER = on;
<QUERY GOES HERE> ;
```

You can set this parameter globally, if needed.

FACTREL_SIZE_THRESHOLD (integer value)

A table with an estimated row count after applying restrictions above this threshold is considered as FACT table by the factrel planner. The default value is 100,000,000.

The following example is an excerpt of a planfile in which the optimizer is choosing all five tables that are involved in the query as FACT because the optimizer-estimated row size (after restrictions) is bigger than the default value of FACTREL_SIZE_THRESHOLD.

```
TABLE SUMMARY (sorted by Snippet/scan order)        # Tables: 5
============================================================
```

Snip	Time	Table Name	Optimizer Estimate	Jit?	-scan Estimate	Table Rowcount	Statistics	Fact?	Distribute On
1	7.087	HOME	2.2B	Jit	---	---	<unknown>	Fact	PLCY_KY
2	23.406	HOME_CVRG	16.6B	Jit	---	17,210,036,918	Full	Fact	PLCY_KY
3	8.190	PLCY_TRM	1.4B	Jit	---	1,481,708,641	Full	Fact	PLCY_KY
4	39.165	HOME_PRM	16.7B	Jit	---	17,210,035,054	Full	Fact	PLCY_KY
5	2739.065	ERN_PRM	71.7B	Jit	---	76,457,753,318	Full	Fact	PLCY_KY

Removing the smallest table (Optimizer Estimate) from the FACT tables can improve the query performance. Here, the PLCY_TRM table is the smallest with 1.4 billion records. You can remove this table from FACT by using following settings:

```
SET ENABLE_FACTREL_PLANNER = on;
SET FACTREL_SIZE_THRESHOLD = 2000000000;
<QUERY GOES HERE> ;
```

If query performance is improved, increase the FACTREL_SIZE_THRESHOLD to higher value (2,500,000,000) to remove the next smallest table (HOME) form FACT and compare the performance.

FACTREL_LIST

The FACTREL_LIST indicates the tables in Netezza that should be excluded from the factrel_size_planner when you are determining whether the table is a FACT table. You can hardcode the tables that you want considered as FACT tables within the optimizer. The FACTREL_LIST ensures the supplied names of tables that the optimizer should tag as FACT and keeps them on the streaming side of the joins if the following conditions are met:

- ENABLE_FACTREL_PLANNER is ON
- Tables meet the FACTREL_SIZE_THRESHOLD requirement

There are instances in which the optimizer can miscalculate a true FACT table and broadcast it across SPUs, which can cause Diskbased Hash Join (DHJ) and eventually run into a performance problem. The following excerpt from `nz_plan` output confirms that DHJ was used:

```
THINGS OF NOTE
==============

Just-In-Time Statistics: 5
Disk Based Join:        2
      5[07]: spu HashJoinNode diskBased=true nParts=91
      5[14]: spu HashJoinNode diskBased=true nParts=68
Cross Product Join:     0
Expression Based Join:   0
Merge Join:             0
MaterializeNode:        0
SUBPLAN_EXPR:           0
Host Distributions:     0

<snip>

      5[07]: spu HashJoinNode diskBased=true nParts=91
                  -- (<Join condition>)
                  -- Cost=90560.2..444254.8 Rows=1.4B Width=77 Size=103.6GB
Conf=15 {(ERN_PRM.PLCY_KY,PRM.PLCY_KY,PLCY_TRM.PLCY_KY)}
                  -- Cardinality ERN_PRM.PLCY_KY 711.5M (Adjusted)
                  -- Cardinality PRM.HOME_KY 33 (Adjusted)
                  -- Cardinality PRM.CVRG_CD 85 (Adjusted)
      5[14]: spu HashJoinNode diskBased=true nParts=68
                  -- (<Join condition>)
                  -- Cost=180693.8..560213.3 Rows=1.4B Width=57 Size=76.7GB
Conf=0
{(HOME.PLCY_KY,HOME_CVRG.PLCY_KY,PLCY_TRM.PLCY_KY,PRM.PLCY_KY,ERN_PRM.PLCY_KY)
}
                  -- Cardinality HOME.HOME_KY 33 (Adjusted)
                  -- Cardinality PRM.HOME_KY 33 (Adjusted)
                  -- Cardinality PRM.CVRG_CD 85 (Adjusted)
                  -- Cardinality HOME_CVRG.HOME_KY 33 (Adjusted)
                  -- Cardinality HOME_CVRG.CVRG_CD 85 (Adjusted)
                  -- Cardinality ERN_PRM.PLCY_KY 691.2M (Adjusted)
```

If you are sure that the broadcast table is a FACT table and it meets the fact table criteria, add that table in the FACTREL_LIST at the session level or globally. Setting FACTREL_LIST at the session level is shown in the following example:

```
SET ENABLE_FACTREL_PLANNER = on;
SET FACTREL_LIST='<table1>, <table2>';
<QUERY GOES HERE> ;>
```

FACTREL_MOVEMENT_COST (integer value)

The default value for this parameter is 1200. Performance suffers when a large fact table is redistributed across SPUs. Performance suffers even more if the fact table is redistributed on a key with low cardinality. There is a good chance that the SPU can run out of temp space. To prevent this condition from occurring, use the FACTREL_MOVEMENT_COST with ENABLE_FACTREL_PLANNER settings.

Consider the following points:

- Recognize the fact tables in the query and mark them as such
- Make the broadcast and distribute operations highly expensive (cost) when the subject is being broadcasted or distributed.
- Both tables in the join contain fact tables, in which case a redistribution is inevitable.

You can make the broadcast and distribute operations highly expensive by using the following settings:

```
SET ENABLE_FACTREL_PLANNER = on;
SET FACTREL_MOVEMENT_COST = 10000;
<QUERY GOES HERE> ;
```

ENABLE_TRANSFORM_JOINEXPR (boolean)

This setting indicates to the optimizer that it should use the join order that is specified in the query instead of evaluating the alternative join order. By default, ENABLE_TRANSFORM_JOINEXPR is set to ON. When set to ON, the optimizer converts ANSI INNER joins that are represented by JoinExpr to FromExpr to open up more combinations for join planning. When set to OFF, INNER joins are processed as specified in the query (as with the OUTER joins). The default setting is ON.

If there is a performance problem with a specific SQL query, check the table join order in the SQL and compare it with the planfile (the "JOIN CONDITIONS" section in the `nz_plan` output). If the join orders are different, disable the ENABLE_TRANSFORM_JOINEXPR and check the query performance. You can try the following setting at session level, but do not set it globally:

```
SET ENABLE_TRANSFORM_JOINEXPR = off;
<QUERY GOES HERE> ;
```

ENABLE_JIT_STATS (boolean)

This parameter enables or disables the JIT-Stats (JIT-Stats are ON by default). The System automatically runs JIT statistics for user tables when it detects the following conditions:

- Tables that contain more than 5,000,000 records.
- Queries that contain at least one column restriction.
- Tables that participate in a join or must have a materialized view that is associated with it.
- The user table is on disk (no external tables, system tables, or virtual tables).

JIT statistics improve selectivity estimations when a table contains data skews or when there are complex column restrictions. The system also uses JIT statistics to avoid broadcasting large tables that were estimated to be small based on available statistics. The overhead of these on-the-fly statistics is negligible when compared to the overall improved query performance and total query time.

Although it is rare, JIT stats can cause overall performance degradation in query execution. If this issue occurs, you must disable JIT statistics for a specific query. You can disable JIT Stats at session level by setting ENABLE_JIT_STATS parameter to OFF or specify a high value for JIT_STATS_MIN_ROWS. The JIT_STATS_MIN_ROWS specifies the minimum number of tuples in a relation (without any restrictions) that are required for JIT-stats to kick in. (The default value is 5000000.) However, never turn off ENABLE_JIT_STATS globally.

ENABLE_2PHASE_PLANNER (boolean)

The ENABLE_2PHASE_PLANNER divides the main plan into two plans: pre-broadcast and main.

The optimizer identifies the small tables in the queries and broadcasts them to join with large tables. The optimizer collects the information about the broadcasted temporary tables at the first stage and uses this information to create a main plan. By default, it is ON.

For certain workloads, the multi-phase planner might not produce an optimal performance. When running a query, it might stop with "ERROR: join_references: variable not in sub-plan target lists".

When your query has many JOINS, the optimizer can choose a plan that is not optimal if it causes you to receive errors that are related to "NOT ENOUGH MEMORY FOR MERGE-STYLE JOIN". In these cases, you must turn off ENABLE_2PHASE_PLANNER. You can use the following setting at the session level, but do not set it globally:

```
SET ENABLE_2PHASE_PLANNER = off;
<QUERY GOES HERE> ;
```

ENABLE_JIT_DISPERSION (boolean)

This parameter enables improved dispersion calculation. By default, it is ON.
There are instances in which JIT stats can cause overall performance degradation in query execution. You must disable JIT statistics for getting dispersion information. When turning ON or OFF dispersion information consider following points:

- A table with more records than the value of JIT_DISP_MIN_ROWS (500M default) tries JIT statistics for getting dispersion information.
- The table includes joins that are unhandled by JIT join
- The table includes aggregates

You must ensure that the other JIT settings match the ENABLE_JIT_STATS, ENABLE_2PHASE_PLANNER, ENABLE_JIT_DISPERSION, and ENABLE_JIT_JOIN settings. If the settings do not match, the file inputs do not line up with the expectations.

INLIST_THRESHOLD (integer value, maximum 100)

An inlist parameter that features a set of literals can be rewritten as a series of equality expressions that are separated by an OR-expression and vice-versa.

For example, `table1.col1 in (1,2,3)` can be rewritten as, `table1.col1 = 1 OR table1.col1 = 2 OR table1.col1 = 3`. In the first case, Netezza creates a transient table and inserts the three values into the transient and a join between table1.col1 and the transient table. A join between tables is costly and the query can take longer to complete. Instead, if OR expressions are used, the optimizer treats them as regular restrictions and the values can be passed down to the FPGA to eliminate of rows more efficiently. For example, if you run the following query with the default INLIST_THRESHOLD setting, that statement makes the query plan ineffective because there is more than five values in the IN clause:

```
SELECT * FROM vtab WHERE tab_id IN (101,203,230,234,250,342,373,414);
```

In this example, you can set the INLIST_THRESHOLD to higher than eight in the session level or globally to improve query performance.

Netezza transforms equal expressions, depending on the number of literals. With the OR-expression, the values can be passed down to the Field Programmable Gate Array (FPGA), which allows for a more efficient elimination of rows.

However, the FPGA has a limit on the number of restrictions it can evaluate. You can use the INLIST_THRESHOLD parameter to cap this transformation. By default, this threshold is a value of five. If an inlist expression that contains less than or equal to five literals is provided in the SQL statement, it is rewritten as a series of equality OR-expressions. If an OR-expression contains more than five literals, it is rewritten as an inlist expression.

The only limitation with large inlist statements is the memory that is needed to compile the query. If the query is too complex, you can run out of memory. However, running out of memory does not affect other sessions because this issue occurs inside the Postgres process that is assigned to that session. The INLIST_THRESHOLD parameter is responsible for this behavior. It is safe to increase this value to 100, but not more.

ENABLE_NEW_CASE (boolean)

The ENABLE_NEW_CASE parameter controls how to handle a CASE statement, as shown in the following CASE statement example:

```
CASE WHEN conditionA THEN valueA
      WHEN conditionB THEN valueB
      WHEN conditionC THEN valueC
      ......
      ELSE valueX
      END
```

If ENABLE_NEW_CASE is set to ON, all conditions and values in the CASE statements are evaluated from the first condition to END and then, the correct value returns according to condition comparison. Netezza must write all comparison clauses as an IF statement in a CPP file; therefore, the CPP file can be large and the compilation time is much longer. Starting with Netezza v.7.0.4 , by default this parameter is ON.

Similarly, If ENABLE_NEW_CASE is OFF, Netezza does not evaluate all conditions. When a condition produces a match, it returns the value and stops. This configuration can make the CPP file simple and reduce the compilation time.

If a query features a large CASE statement, the compilation time can significantly affect the query performance time.

If you encounter a query that is running slow after Netezza upgrade to v.7.0.4 or higher, check for the following conditions:

- Query having CASE statement
- A large CPP file was generated

If both these conditions are true, you can disable the ENABLE_NEW_CASE and attempt the query.

Conclusion

This web doc described the following common Postgres configuration parameters that you might want to check to improve query performance:

- ENABLE_MERGEJOIN
- ENABLE_FACTREL_PLANNER
- FACTREL_SIZE_THRESHOLD
- FACTREL_LIST
- FACTREL_MOVEMENT_COST
- ENABLE_TRANSFORM_JOINEXPR
- ENABLE_JIT_STATS
- ENABLE_2PHASE_PLANNER
- ENABLE_JIT_DISPERSION
- INLIST_THRESHOLD
- ENABLE_NEW_CASE

Before you decide to set any of these parameters globally, check the query and system performance by setting the parameter at the session level.

Related information

For more information, see the following resources:

- IBM PureData System for Analytics (powered by Netezza technology) IBM Knowledge Center
  https://www.ibm.com/support/knowledgecenter/SSULQD/SSULQD_welcome.html

- IBM PureData System
  http://www-01.ibm.com/software/data/puredata/analytics/

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided “AS IS”, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

This document was created or updated on June 23, 2016.

Send us your comments in one of the following ways:

- Use the online **Contact us** review form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

This document is available online at http://www.ibm.com/redbooks/abstracts/tips1341.html .

Trademarks