Best Practices for Writing and Tuning SQL

V1.11

Mark Mierzejewski
Robert Smith
Teradata Professional Services

October 2013

TERADATA

Agenda

- Best Practices for Writing SQL
- Best Practices for Tuning SQL
- SQL/Tuning Checklist
- Tuning Examples

Best Practices for Writing SQL

TERADATA

Best Practices for Writing SQL

- A Matter of Style
- Know Your Data
- Building your Query
- Indices A.K.A. Indexes
- Partitioned Primary Index
- Access Method Comparison Summary
- Index Hashing
- Will The index Be Used?
- What Do You Really Need?
- Allocation of Spool Space
- Skewing
- Exploring data
- Creative SQL Options
- Volatile Tables

A Matter of Style

- Change proprietary keyword shortcuts (sel, ins, upd, ct, etc) to their full word equivalents (SELECT, INSERT, UPDATE, CREATE TABLE,etc)
- Convert SQL keywords (particularly in FROM clause) to upper case
- Build a block structure where each item of a clause begins in same column.
- Avoid using the comma-separated implicit join syntax. The explicit ANSI syntax makes joins easier to read and makes it much less likely that join conditions will be forgotten:

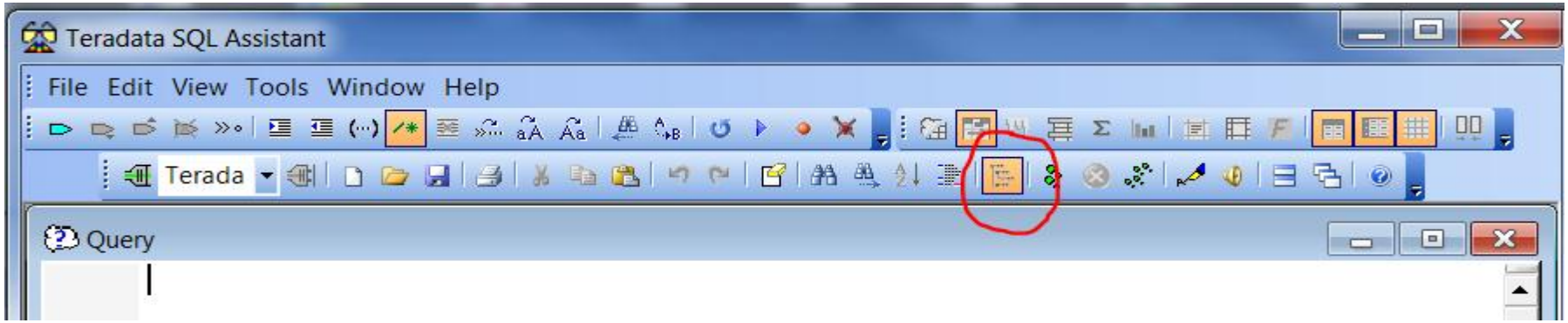
  . . . ~~From Tbl1, Tb2 Where Tb1.col1 = Tbl2.col1~~ ← not so good

  . . . From Tb1 Inner Join Tb2 On Tb1.col1 = Tb2.col2 ← better

- Indent for sub-queries
- Structure code to allow quick cut & paste for diagnostic efforts
- Break long lines up (don't let end of the line flow beyond the edge of a reasonably sized edit window)

A Matter of Style (Cont.)

- When breaking up list items – put items a separate lines with comma preceding
- Eliminate blank lines, do not overdo white space. Try to make the SQL compact to display a reasonable amount of logic
- Comment your code accurately and keep comments up to date.
- Basically make it readable and understandable!
- SQL Assistant has a basic formatter Menu: Edit → then Format Query, or the toolbar button below.

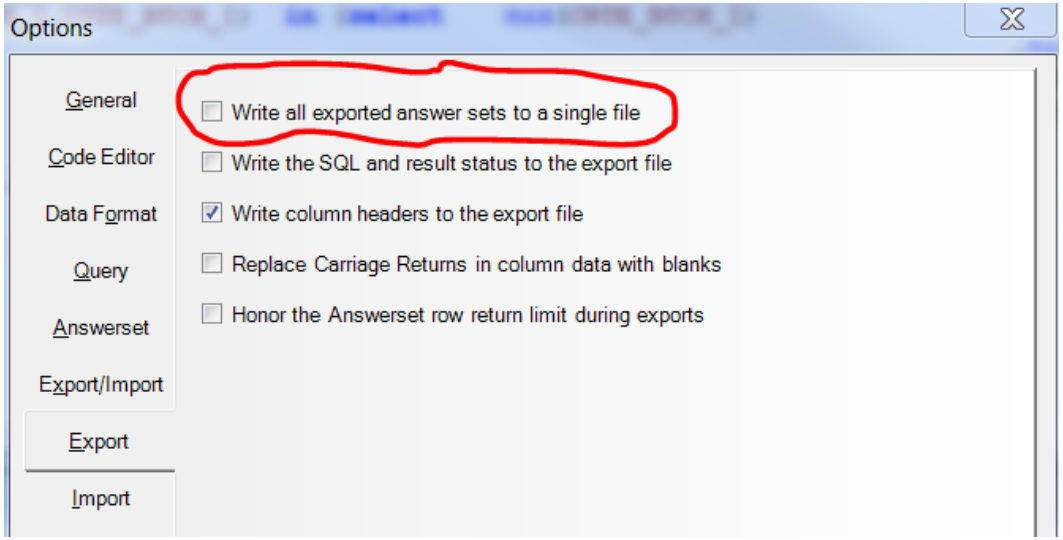


Know your data

- Know your data. Familiarize yourself with indices, statistics and type of data in the tables. Can use SQLA commands to gather this info.
  - **Show View database.viewname** – Shows view definition DDL including source tables referenced and any included business logic.
  - **Show Table database.tablename** – Shows the table definition DDL including data-types, defined indices (primary and secondary)
  - **Help Statistics database.tablename –** Shows which columns on the table have had statistics collected, the collection date and time, and the count of occurrences. If a view is a cover view, you can also execute "help statistics database.viewname."
  - **Help Index database.tablename –** Shows the indexes defined for a table. If a view is a cover view, you can also execute "help index database.viewname."

Know your data (Cont.)

- Place a Show in front of your select statement. This will show all the underlying views and tables referenced by your query.
  - Will want to run this as an export Select File → Export.
  - To send to a single file: Tools → Options → Export. Check the box "Write all exported answer sets to a single file"



- Can also query against DBC views. (DBC.Tables, DBC.Indices, DBC.Columns)

Building your query

- EXPLAIN your SQL and tune SQL accordingly
  - In SQL Assistant, prefix your query with keyword “Explain” or select the query and press “F6”
  - If estimate is in hours , days, or years - don't run it! Tune it!
- Start with a trivially small set of data to validate the SQL. It’s easier to spot errors and data issues, and there’s little to gain by tuning incorrect SQL. Your tests will also run faster.
- Use equality, “In”, and “And” conditions if possible, Avoid inequality, “Like”, and “Or” conditions
- Build your query iteratively, adding new pieces each time. Run Explain at each stage to immediately catch when something potentially bad has been added.
- Queries should be written against views (which should already include “locking row for access” clause) or if queries need to be written against tables directly, the “locking row for access” clause should be included.

Indices A.K.A. Indexes.

- Always try to reference a table via an index if possible. - Add qualifiers to "where" clauses and "on" clauses against index fields.
- When using a PPI (Partitioned Primary Index) always limit the partition field with a lower and upper bound. This allows the optimizer to utilize partition elimination.
- If the table has secondary indexes, having statistics collected on the index columns let's the optimizer know if it should use the secondary index.
- If using non-index columns in joins or constraints, check that statistics are collected on those columns (if not, validate performance with and without statistics and request additional statistics if needed).

Partition Primary Index (PPI)

- Conceptually, think of it as a single virtual table that is partitioned into N different smaller tables
  - Rows are still hash distributed among the AMPs on the primary index columns
  - Rows are ordered by partition, then by Primary Index Hash within partition
- Partitioning column can be part of primary index, but is not required to be
- In many cases, better performance occurs when partitioning column is part of the primary index
- Maximum of $9.2*10^{18}$ partitions and 62 levels (65,535 partitions and 15 levels prior to Teradata 14.0), numbered from one
- One or more columns in partitioning expression

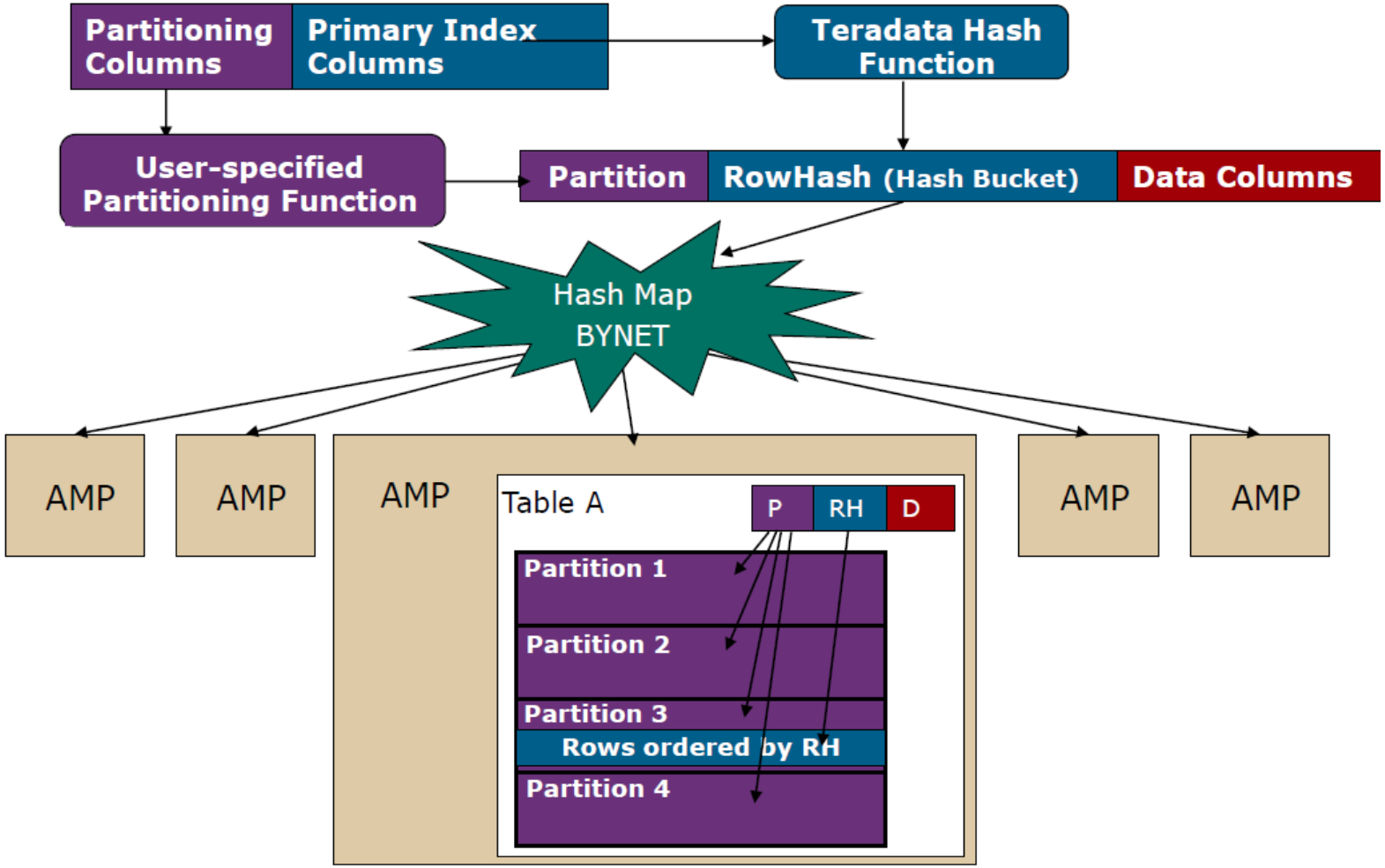
Partition Primary Index (PPI) (Cont.)

- For PI to be unique (UPI with partition), partitioning column must be part of the PI
- No character or graphic comparison allowed in partitioning expression

- Possible degradation of PI Access
  - If partitioning column is not qualified, all partitions will be read
  - Joins on PI columns from a non-PPI table to a PPI table will result in comparing PI column in every PPI table partition

- Important to collect stats the partition column and the partition keyword.
  - `COLLECT STATISTICS on MY_TABLE COLUMN (BUSINESS_DT);`
  - `COLLECT STATISTICS on MY_TABLE COLUMN (PARTITION);`

Partition Primary Index (PPI) (Cont.)



Access Method Comparison Summary

Unique Primary Index

- Very efficient
- One AMP, one row
- No spool file

Non-Unique Primary Index

- Efficient if the number of rows per value is reasonable and there are no severe spikes.
- One AMP, multiple rows
- Spool file if needed

No Primary Index

- Access is a full table scan without secondary indexes.

Unique Secondary Index

- Very efficient
- Two AMPs, one row
- No spool file

Non-Unique Secondary Index

- Efficient only if the number of rows accessed is a small percentage of the total data rows in the table.
- All AMPs, multiple rows
- Spool file if needed

Partition Scan

- Efficient since because of partition elimination.
- All AMPs; all rows in specific partitions

Full-Table Scan

- Efficient since each row is touched only once.
- All AMPs, all rows
- Spool file may equal the table in size

The Optimizer chooses the fastest access method on a base table or a join index.

COLLECT STATISTICS to help the Optimizer make good decisions.

Index Hashing

The hashing algorithm generates a row id from the value(s) of the indexed column(s).

- The order of the column(s) is not significant:

  HashRow(a,b) = HashRow(b, a)

- All columns are needed to use the index. If index is (a, b, c):

  "where a = 1 and b = 2" requires an all-rows scan

- To use hashed indexes, the index column(s) must be present in an equality constraint. If index is (a, b, c):

  "where a = 1 and b = 2 and c > 3" requires an all-rows scan

- All integer types (ByteInt, SmallInt, Integer, BigInt) hash the same
- Decimal types (with only an integer component) hash the same as integer types
- Float columns hash differently from other numeric types
- Character data types CHAR and VARCHAR hash the same regardless of length. Trailing spaces are insignificant, as are capitalization. Starting with R13.x, character set (Latin, Unicode) is also insignificant

Will the Index Be Used?

```
Y CharColumn = 'ABC'
Y CharColumn = (('ABC'(char(5)))||'DEF')
N CharColumn = 3   ←Char implicitly converted to numeric

Y IntegerColumn = '3'   ←Char implicitly converted to numeric
Y IntegerColumn = 3.14159*(3**2)
N IntegerColumn = UDF_Function(1)
Y IntegerColumn = EXTRACT(YEAR  FROM CURRENT_DATE)*100
                + EXTRACT(MONTH FROM CURRENT_DATE)
? IntegerColumn BETWEEN 12345 AND 12350 ←Depends upon stats,
  only applies to value-ordered secondary and hash indexes

Y DateColumn = DATE - 3 ←Index used when colm = const_expr
N DateColumn + 3 = DATE   ←Index NOT used when colm_expr=const
Y DateColumn = '2003-10-03'   ←ANSI date always works
Y DateColumn = 101
? DateColumn = '10/03/03'   ←Depends upon DATEFORM setting
```

What Do You Really Need?

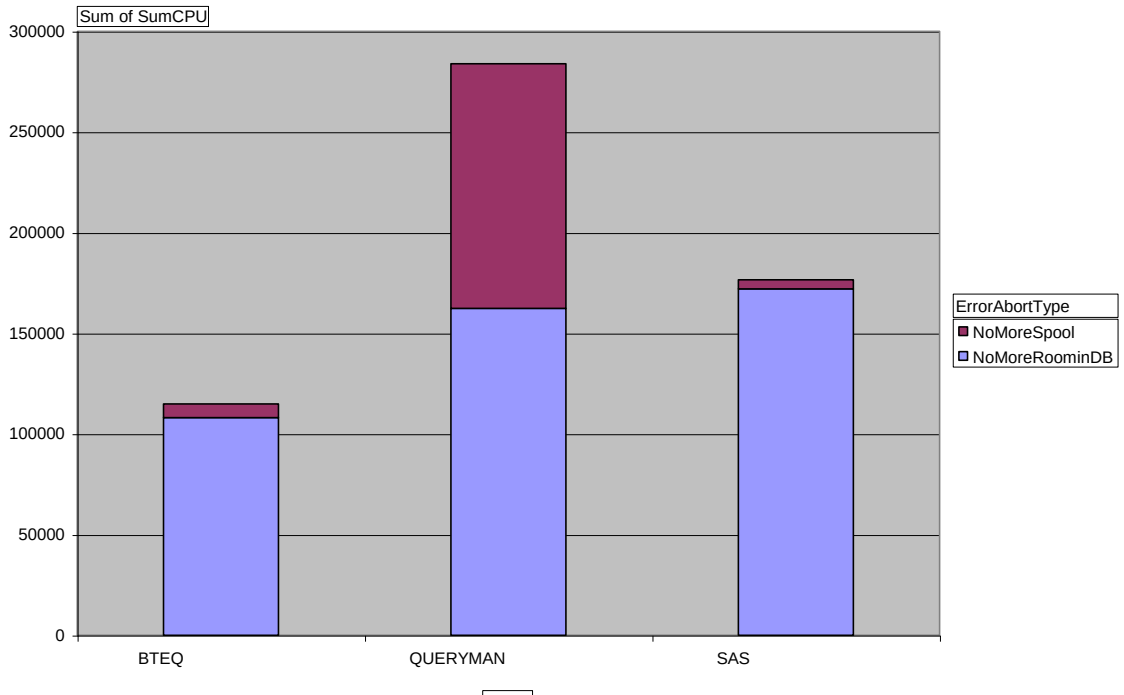
- In any SELECT in the primary SQL or derived table SQL, don't bring in a column unless you need it. (Avoid Select *)
- Don't bring in a row unless you need it (qualify in “where” clause). Look at all value constraints to see how quickly the working set of rows can be reduced.
- If using functions (Coalesce, Trim, concatenation, etc.) on terms in the select list or the join conditions, use caution. This impacts the potential use of indices and statistics.

Allocation of Spool Space

- Spool space is used for intermediate storage while processing an SQL request. It is also used to store volatile tables
- Spool can be assigned directly to a user or through the user's profile
- All concurrent requests by a user share the same space (so the limit is effectively reduced – first-come, first served).
- Spool space is allocated at the VAmp level…
  - **Select** 3*10**12 **as** spool_limit
    ,HashAmp()+1 **as** #_VAmps
    ,spool_limit/#_VAmps **as** VAmp_spool_limit
  - **Select** HashAmp()+1 = 1152
  - 3,000,000,000,000/1152 = 2,604,166,666
- …so you can run out of spool even though the current sum of your spool is less than the total limit (i.e. one or more VAmps have reached the limit).

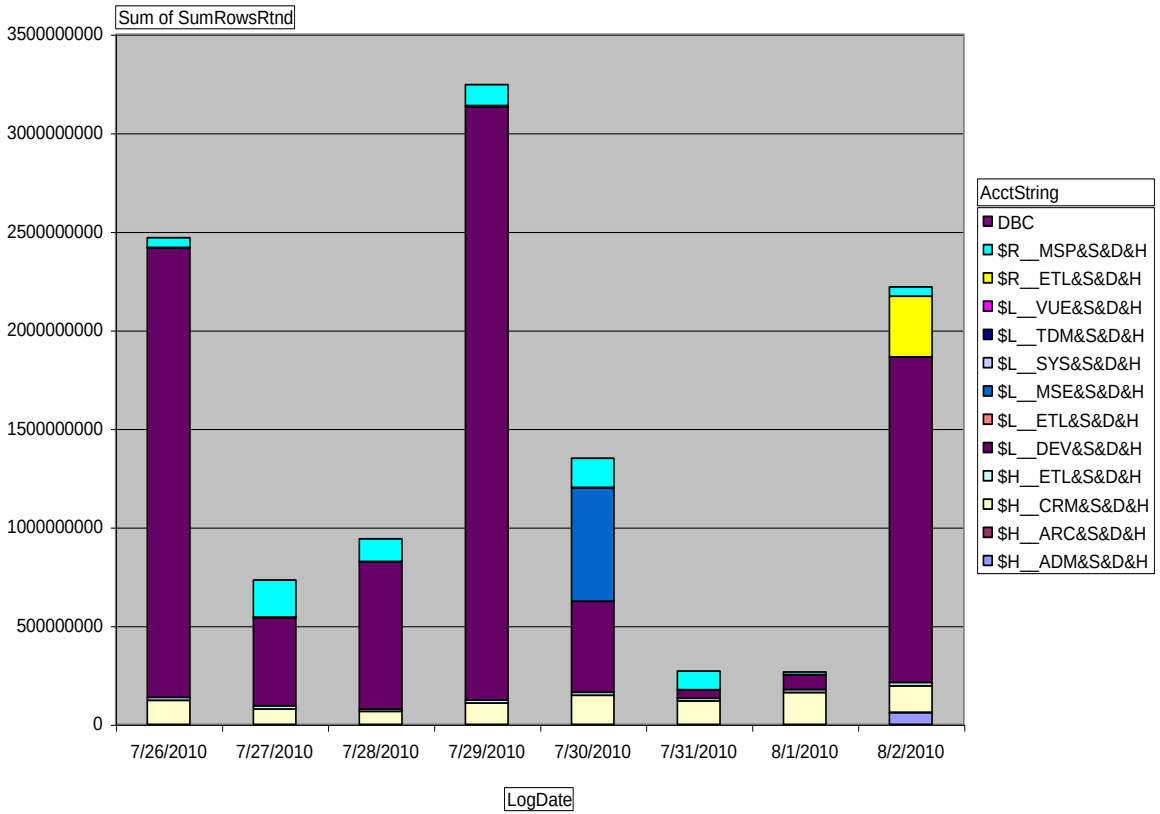
Skewing – "No More Spool" and "No More Room in Database" Errors

- The Errorcode and ErrorText columns in DBQL can also provide Performance Information
- DBQL and Acctg does not accurately reflect CPU and IO numbers for failed Requests so The Skew volume in the previous slide does not accurately represent these queries.
- Also, the Chart below does not accurately reflect all the CPU that's wasted by these failed requests, but it's a start



Exploring Data -Avoid "Select * From…" when just wanting to "see what the data looks like"

- From Doing this Analysis at the AccountString level, it allowed us to easily identify the Development group selecting a lot of rows
- By Looking at the Actual Queries Logged, we've identified that they are Numerous Large Select * From… Statements
- Calling a couple of Users, produced the answer "we just want to see what the data looks like".



	Without Sample	With Sample	Sel Top 2000 *
Rows	2,579,781,818.00	2,000.00	2,000.00
Spool	253,013,456,896.00	513,024.00	228,864.00
Time Ran	03:26.4	0:10.01	0:22.02
Total CPU	1,732.04	9.63	0.35
total I/O	4,672,244.00	23,762.00	33

Analysis shows using "Top" is even more efficient than Sample. Note the CPU and IO differences.

TERADATA

Creative SQL Options

- Avoid the use of EXCEPT/MINUS. Alternatives available are:
  - Left join with null check in where clause
  - Use "where not exists" against a co-related sub-query.
- Derived Tables are useful to prepare data in-line with an existing query
  - A derived table is simply a select statement enclosed in parentheses and given an alias:

    `. . . FROM (<select statement>) Tb1 . . .`
  - Pay attention to the order of the field selection, try to match the order of fields in indices of physical table that it joins to (or the selection fields of another derived table that it joins to).
  - Make the dataset as small as possible (Rows and Columns).
  - Exist only for the life of the query
  - Caution: consumes user's spool space.
- Volatile Tables are a good alternative to replace Derived tables.
  - Volatile table enables creation of primary and secondary indices
  - Allows Statistics collection
  - Exist for the life of the session.
  - Caution: consumes user's spool space.

Volatile Tables

- Specify as a multi-set table – to eliminate costly duplicate checks
- Choose PI for access -match PI of table(s) you will be joining to (match the biggest table first)
- Remember the "on commit preserve rows"; otherwise table will be empty
- Two Methods to create:
  - Create table and then insert into it
  - Create table on the fly using a select - requires "with data" option does not tell you how many rows are loaded to the table)
- Repetitive query patterns (typically found in derived tables) are good candidates for becoming Volatile Tables
- Collect statistics on PI and any other joined fields.
- Create via Select Syntax:

```
CREATE VOLATILE MULTISET TABLE  VT_MYTABLE AS
                (SELECT CURRENT_DATE AS MY_DATE
        ) WITH DATA
  PRIMARY INDEX (MY_DATE)
       ON COMMIT PRESERVE ROWS;
```

Best Practices for Tuning SQL

TERADATA

Best Practices for Tuning SQL

- Preparing to Tune Already Written Queries
- Using the DBQL to Drive the Tuning
- Using DBQL Step Data
- Explain, Explain, Explain
- A Case For Statistics Collection
- Collect Statistics Guidelines
- DBQL Basics
- Measure the Affects of Tuning Efforts.
- Measuring Query Performance
- Using the DBQL Macro
- Sample DBQL Macro Output

Preparing to Tune Already Written Queries

- Keep a copy of the original query for reference.
- If the query is not easily readable, re-format the SQL being tuned to make it readable without changing the structure and logic of the SQL.
- Resolve unqualified references to columns with the table alias.
- Keep a log of all changes made, link them to performance information collected during the period of the change.
- The DBQL has extensive information that can help identify candidates for tuning.
- The Performance Indicators from DBQL
  - Product Joins - Too Much CPU compared to IOs (PJI)
  - Large Scans - Too Much IO with Little CPU (UII)
  - CPU or IO Skew (CPUSkw or IOSkw)
  - High CPU or IO
  - Impact CPU (a measure of the Impact of Skewing)
  - Significant Response Time

Using the DBQL to Drive the Tuning

- From the DBQLStepTbl_Hst view, start with the worst step. i.e. high CPUTime, IOCount, SpoolUsage and/or long-running step (StepStopTime minus StepStartTime). You can use the EXPLAIN for reference as well.

- Beware the sliding window merge join step. It may look good at first, but can be very costly.

- Pull from DBQL a sample set of the SQL execution in the last x number of days (i.e. week), and steps for one execution (specific SQL being tuned having identified symptoms). Assuming you have the QueryId, (and ProcId and LogDate/run date of the SQL to tune) you can run the following SQL against DBQL:

```
SELECT * FROM pdcrinfo.DBQLStepTbl_Hst
WHERE  QueryId=163617508142966885
-- optional      AND  ProcId=16361
-- optional      AND Logdate='2013-09-03'
-- optional      AND SQLtext LIKE '%SELECT T1.F1,%'
```

Using DBQL Step Data

- Each step contains start and finish times, resources used, and all the Performance Indicators can be calculated for each step
  - Look at differences between estimated row count and actual for each step
  - Can show blocking (not what was blocked)
  - An "All amp" operation taking a long time even though it is not using much resources, may indicate AWT shortage
  - Query taking long in aggregation step may indicate aggregate join index will help
- Use to identify plan changes when moving to New Releases or other System Changes have occurred.
  - Can be done by # of steps or type of steps

Explain, Explain, Explain

- Running the Explain is the most useful tool in a Developer's/Tuner's toolbox.
- To use in SQL assistant, press "F6" with the query selected that you want to check or simply prefix your query with the Keyword "Explain" and execute the query.
- Explains are reviewed for the following items:
  - Confirm estimates are reasonable
  - Missing joins (implicit joins) and join constraints
  - Data format issues (i.e. character to integer)
  - Mis-matched aliases
  - CASTing errors
  - Unnecessary PRODUCT JOINS
  - Look for "No Confidence" from optimizer output
  - Look at the order in which tables are joined
  - Look at join types (merge, partition, hash, etc.)
  - Large redistributions (moving rows in preparation for a join)

Explain, Explain, Explain (Cont.)

Symptom	Probable Root Cause
Long running merge step	Bad primary index (skewed and/or costly duplicate row checking) Secondary indexes Update of an index column
Long running join steps with TRANSLATE functions	Unmatched data types
Large re-distributions	Missing join conditions Join conditions don't match primary index Unmatched data types on join conditions
Skewed re-distributions	Obsolete or missing statistics (the optimizer failed to recognize a skewed distribution) Unmatched data types Columns from the INNER table of an OUTER join are used in a subsequent join condition
Resource intensive aggregations	Complex CASE logic in the GROUP BY clause High cardinality on a very large data set
Runaway product joins and/or large duplications	Obsolete or missing statistics Unmatched data types on join conditions OR-ed join conditions Unequal join conditions A large number of predicates resulting in in underestimated row counts Functions (substring, trim, like, upper) in join conditions and predicates

Explain, Explain, Explain (Cont.)

- Use Explain liberally, there is minimal impact to the system from running explains.
- Things to look for in an explain plan that you wish you did not see!

```
size of Spool 37 is estimated with no confidence to be *** rows (
*** bytes). The estimated time for this step is 174,474,440 hours and 4 minutes
```

```
We do an all-AMPs RETRIEVE step from PRODPURPLE.PMTN_ITEM_STR
          by way of an all-rows scan with no residual conditions
```

```
with a join condition of ("(ACTV_F = ACTV_F) AND ((ACTV_F = (TRANSLATE((ACTV_F )USING
LATIN_TO_UNICODE))) AND ((DEPT_I = MDSE_DEPT_REF_I) AND ((ACTV_F = (TRANSLATE((ACTV_F)USING
LATIN_TO_UNICODE))) AND ((VEND_I = VEND_I) AND ((DEPT_I = DEPT_I) AND ((PO_I = PO_I) AND
((PO_NUM_SRC_C = PO_NUM_SOURCE_C) AND ((PO_APRO_D = (CAST(({RightTable}.AUD_CRTE_TS) AS DATE)))
AND((VEND_REF_I = VEND_I) AND ((ACTV_F = (TRANSLATE((ACTV_F )USING LATIN_TO_UNICODE))) AND
((ACTV_F = (TRANSLATE((ACTV_F )USING LATIN_TO_UNICODE))) AND ((ACTV_F = ACTV_F) AND (DEL_F =
DEL_F))))))))))))").
```

```
which is redistributed by the hash code of (PRODRPT.WK_MDSE_DSPL_STR_ITEM.BRCK_MRTR_STR_I,
     PRODPRS.MDSE_DSPL_ITEM.MDSE_ITEM_I) to all AMPs.  The result spool file will not be cached
in memory.  The size of Spool 22 is estimated with no confidence to be 783,989,854 rows (
     30,575,604,306 bytes).  The estimated time for this step is 12.19 seconds.
```

A Case for Statistics Collection

- The Teradata optimizer is a cost-based optimizer that relies on accurate table statistics.

- Statistics problems can lead to Performance Problems
  - **Redistributions** of large tables and spools
  - **Product Joins** on large numbers of rows
    - Heavy CPU consumption by a process at the expense of all other users on the system
  - Aggregates early in a plan on large numbers of rows
  - **Skewed** distribution of spool tables
    - Reduces parallel efficiency

- Collecting statistics on appropriate objects improves optimizer cost estimates

- Performance improvements of more than 90% can be achieved by collecting proper statistics!

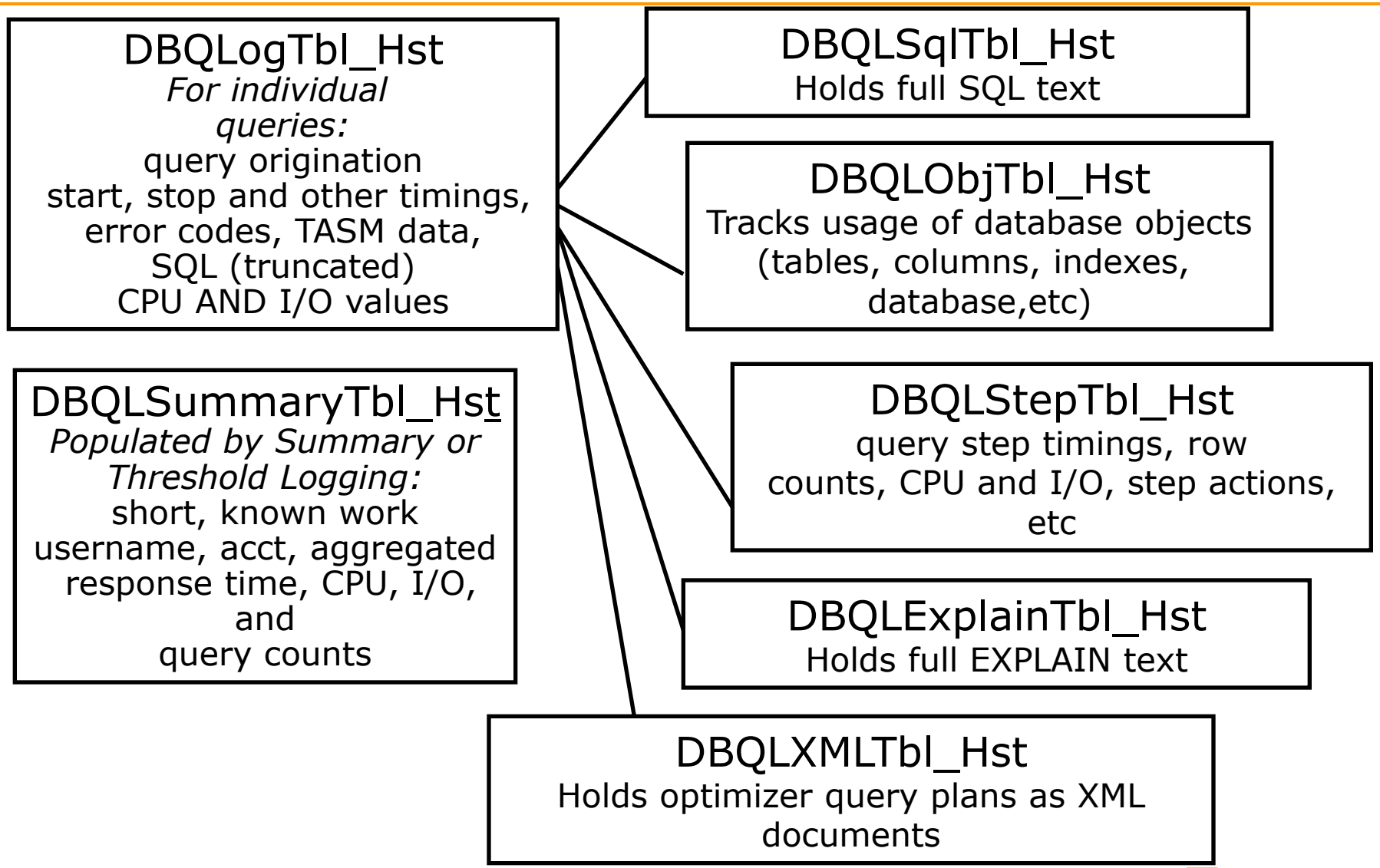
Collect Statistics Guidelines

- Stale or incorrect statistics
  - Stale Statistics can be more damaging the no statistics at all
  - **Old statistics** may be providing false information to the optimizer – Recollect statistics that may be stale
  - Look for **inaccurate statistics** that were collected when the table had zero rows. This can occur when statistics are collected prior to the insert of rows rather than after

- Do not waste CPU with over-collection
  - Time the statistics collections - Statistics should be recollected for existing tables when the population of a column changes by > 10% (rule of thumb)
  - Do not collect on columns that are not used in the WHERE clause
  - Note that Collecting Statistics on columns in the WHERE and ON Clause that are aggregated, concatenated, sub-stringed, etc. will provide no value

- Do Not collect on every recommendation provided by Diagnostic HelpStats or the Statistics Wizard

DBQL Basics – The Views in PDCRINFO DB



Measure the Effects of Tuning Efforts

- Solutions should be practical i.e. Don't change a primary index on a major table if statistics collection will solve the problem.
- Use DBQL Data to Compare Performance Measurements

Example Showing the Impact of Tuning a Significantly Skewed Query

	Current Request	Request with Tuning
UserName	BI_ETL_Usr	Tuner_id1
Frequency	Every Day	
StartTimeStamp for 1 occurrence	2009-03-11 11:59:04.25	2009-04-08 10:47:28.42
Duration	4 Hours 41 Minutes	1 Minute 7 Seconds (PSF Priority is 4X less than production id)
CPU Consumption	13,351 CPU Seconds	293 CPU Seconds
IO Count	17,461,158	1,117,554
CPU Skew	46.44	1.05
ImpactCPU	288,842	308

Measuring Query Performance – MyQrys Macro

This macro is intended to make it easy for Teradata users to see the performance characteristics of their (or other's) SQL requests. It pulls information for either the current (DBC) tables, the history (PDCRINFO) tables, or both, depending on the date range supplied. It uses a "Where" condition on both parts of the union to eliminate accessing unneeded current or history tables depending on the from/to dates.

Syntax:

exec pdcrinfo.MyQrys ([User_Id], [From_Date], [To_Date], [Query_Fragment]);

Parameters:

- **User_Id** defaults to the current user
- **From_Date** defaults to Current_Date
- **To_Date** defaults to Current_Date (Dates are in the format 'yyyy-mm-dd')
- **Query_Fragment** – defaults to Null. If supplied, only rows where SQLTextInfo contains the specified string are returned. The maximum string length is 100 characters.

Measuring Query Performance – MyQrys Macro (Cont.)

Note that all four parameters are optional, but must be designated by commas when omitted; i.e. the parenthetical list must always contain 3 commas.

Examples:

- **exec MyQrys(,,,);**

  Returns all requests submitted today by the current user. This query accesses only the DBC tables.

- **exec MyQrys(,Current_Date-7,,);**

  Returns all requests submitted by the current user in the past week (including today). This query accesses both DBC and PDCRINFO tables.

- **exec MyQrys('a524897', '2013-07-04', '2013-07-05', 'PO_LOC_SHIP_VIOL_FCT');**

  Returns all requests submitted on July 4th and 5th by user a524897 that contain the string 'PO_LOC_SHIP_VIOL_FCT'. This query accesses only the PDCRINFO tables.

Sample DBQL Macro Output

UserName	LogDate	SessionID	ProcID	QueryID	AcctString	ExpendAcc	DefaultDat	SpoolUsage	StartTime	TotalRespTime
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:29 PM	0 00:00:00.030000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:29 PM	0 00:00:00.130000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:30 PM	0 00:00:00.070000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	6,500,127,232.00	9/17/13 2:30 PM	0 00:06:59.600000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:30 PM	0 00:00:00.290000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	1,769,472.00	9/17/13 2:29 PM	0 00:00:01.650000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	82,575,360.00	9/17/13 2:22 PM	0 00:00:01.290000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:22 PM	0 00:00:00.110000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	1,769,472.00	9/17/13 2:29 PM	0 00:00:04.860000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:29 PM	0 00:00:01.710000
Z046438	9/17/2013	24,440,875	16,371	163,717,508,144,198,000	BDQ00001&S&	BDQ00001	Z046438	0	9/17/13 2:29 PM	0 00:00:00.170000

TDWMDelay	DelayTime	NumOfActiv	NumResul	SumCPU	SumIO	CpuSkw	IOSkw	PJI	UII	ImpactCPU
<null>	0 00:00:0	0	0	0	0	0	0	0	0	0
<null>	0 00:00:0	1	107	0	8	0	1,152.00	0	0	0
<null>	0 00:00:0	0	0	0	0	0	0	0	0	0
<null>	0 00:00:0	1,152	52	2,527.31	38,039,568.00	1.27	1.18	0.07	15.05	3,211.78
<null>	0 00:00:0	0	0	0	0	0	0	0	0	0
<null>	0 00:00:0	1,152	1	3.12	26,532.00	4.44	1.17	0.12	8.5	13.82
<null>	0 00:00:0	1,152	12,078.00	5.14	36,876.00	4.48	1.03	0.14	7.17	23.04
<null>	0 00:00:0	0	0	0	0	0	0	0	0	0
<null>	0 00:00:0	1,152	1	2.92	31,140.00	4.74	1.15	0.09	10.68	13.82
<null>	0 00:00:0	0	0	0	0	0	0	0	0	0
<null>	0 00:00:0	1	107	0	8	0	1,152.00	0	0	0

QueryText
HELP COLUMN "PRODRPT_V"."MDSE_ITEM_DIM_V".*
HELP TABLE "PRODRPT_V"."MDSE_ITEM_DIM_V"
SELECT MDSE_ITEM_DESC_T , SUM (EXT_SLS_PRC_A) FROM PRODRPT_V.MDSE_SLS_AGG_V SLS LEFT JOIN PRODRPT_V.MDSE_ITEM_DIM_V DIM ON MDSE_ITEM_I = MDSE_I
SELECT MDSE_ITEM_DESC_T , SUM (EXT_SLS_PRC_A) FROM PRODRPT_V.MDSE_SLS_AGG_V SLS LEFT JOIN PRODRPT_V.MDSE_ITEM_DIM_V DIM ON SLS.MDSE_ITEM_I = DIM
SELECT MDSE_ITEM_DESC_T , SUM (EXT_SLS_PRC_A) FROM PRODRPT_V.MDSE_SLS_AGG_V SLS LEFT JOIN PRODRPT_V.MDSE_ITEM_DIM_V DIM ON SLS.MDSE_ITEM_I = DIM
SELECT TRIM(DATABASENAME),TRIM(TABLENAME) FROM DBC.TABLESV WHERE UPPER(TRIM(DATABASENAME)) LIKE UPPER(TRIM('PRODRPT_V')) ESCAPE '\' AND UPPER(TRIM
SELECT NULL(VARCHAR(512)),DATABASENAME(VARCHAR(512)),TRIM(TABLENAME(VARCHAR(512))),CASE WHEN DATABASENAME <> 'DBC' THEN CASE TABLEKIND WHEN 'T' T
HELP SESSION
SELECT TRIM(DATABASENAME),TRIM(TABLENAME) FROM DBC.TABLESV WHERE UPPER(TRIM(DATABASENAME)) LIKE UPPER(TRIM('PRODRPT_V')) ESCAPE '\' AND UPPER(TRIM
HELP COLUMN "PRODRPT_V"."MDSE_ITEM_DIM_V".*
HELP TABLE "PRODRPT_V"."MDSE_ITEM_DIM_V"

SQL/Tuning Checklist

Tuning Checklist

1. Ensure that Statistics are collected and up to date on columns involved on joins.

```
COLLECT STATISTICS on MY_TABLE COLUMNS (ORDER_NUM);
```

2. Eliminate Full table scans – Join on indices. (Identify using Explain)

3. Eliminate Product joins With High estimates – These are rarely good and indicate missing join criteria. (Identify with Explain) or missing stats.

4. If PPI's are available, you want to leverage Dynamic Partition Elimination's or see "x" number of partitions accessed. (Don't leave the partition unbounded on either end).

5. Eliminate Translates – implicit type conversion (Identify using Explain).

Tuning Checklist (Cont.)

6. Avoid use of Except/Minus, use left join or where not exists instead.

7. Avoid use of Select * (increases use of Spool), explicitly select fields. One Exception is select top 10 *for data interrogation.

8. Qualify rows as early in the query as possible. This includes derived and volatile tables as well. (where clauses)

9. Leverage Volatile tables for Repetitive passes against data. Create a volatile table for queries that are run multiple times as derived tables. i.e.

```
Select max(tablename.datefield)
from tablename;
```

10. Use functions (cast, coalesce, case, trim) on the smallest set of data possible. (i.e. after data has been filtered or joined.)

Tuning Example #1 Statistics

Tuning Example #1 - Statistics

```
Select a.SVCRQ_ACTV_I, a.SVCRQ_SYS_I, c.CODE_C
     , max(ACTV_UPDT_TS) as ACTV_UPDT_TS
from (select a.SVCRQ_SYS_I, a.SVCRQ_ACTV_I, a.ACTV_UPDT_TS
         , a.ACTV_STRT_TS, SVCRQ_ACTV_TYPE_DIM_I
      from prod_v.SVCRQ_ACTV a
        ,(select   SVCRQ_SYS_I, SVCRQ_ACTV_I
                 , max( ACTV_UPDT_TS) as ACTV_UPDT_TS
          from prod_v.SVCRQ_ACTV
          where SVCRQ_ACTV_TYPE_DIM_I=7144
            and cast(ACTV_STRT_TS as date) is not null
          group by SVCRQ_SYS_I, SVCRQ_ACTV_I
         ) ,(select SVCRQ_SYS_I, min( ACTV_end_TS) as ACTV_end_TS
              ,ACTV_STRT_TS
          from     prod_v.SVCRQ_ACTV
          where    SVCRQ_ACTV_TYPE_DIM_I=7144
            and    cast(ACTV_STRT_TS as date) is not null
          group    by SVCRQ_SYS_I, ACTV_STRT_TS) c
        ,(select SVCRQ_SYS_I, min(ACTV_STRT_TS) as ACTV_STRT_TS
          from     prod_v.SVCRQ_ACTV
          where SVCRQ_ACTV_TYPE_DIM_I=7144
            and cast(ACTV_STRT_TS as date) is not null
          group    by SVCRQ_SYS_I) d
Where a.SVCRQ_SYS_I=b.SVCRQ_SYS_I
   and a.ACTV_UPDT_TS=b.ACTV_UPDT_TS
   and a.SVCRQ_ACTV_I=b.SVCRQ_ACTV_I
   and a.SVCRQ_SYS_I=c.SVCRQ_SYS_I
   and a.ACTV_end_TS=c.ACTV_end_TS
   and a.ACTV_STRT_TS=c.ACTV_STRT_TS
   and a.SVCRQ_SYS_I=d.SVCRQ_SYS_I
   and a.ACTV_STRT_TS=d.ACTV_STRT_TS) a,
(select distinct SVCRQ_SYS_I
  from prod_v.cctr_svcrq b
  where cast(crte_ts as date)>='2013-08-01'
     and cast(crte_ts as date)<='2013-08-21') b
, prodrpt_v.CODE_DIM c
Where a.SVCRQ_SYS_I=b.SVCRQ_SYS_I
   and a.SVCRQ_ACTV_TYPE_DIM_I = c.CODE_DIM_I
   and c.CODE_DIM_I in (7144)
Group by a.SVCRQ_ACTV_I,  a.SVCRQ_SYS_I , c.code_c;
```

Tuning Example #1 - Statistics

StartTime	UserName	NumResultRows	AMPCPUTime	TotalIOCount	impactCPU
8/23/2013 14:39:13.27	S002823	20,942.00	25,396.71	50,592,868.00	29,153,129.47
8/24/2013 15:01:26.36	S002823	21,025.00	22,317.56	50,694,499.00	25,621,009.92

"helpstats" suggested collecting stats on two columns. Of these, SVCRQ_ACTV.SVCRQ_ACTV_TYPE_DIM_I had significant skewing:

SVCRQ_ACT	Count(*)
7145	8194803
7144	2854717
7147	2460838
7511	333778
7510	168623
8241	52784
. . .	. . .

COLLECT STATISTICS COLUMN SVCRQ_ACTV_TYPE_DIM_I ON SANDBOX_PS_DBA.SVCRQ_ACTV ;

This statistic made the optimizer aware of the skew and resulted in a different plan.

StartTime	UserName	NumResultRows	AMPCPUTime	TotalIOCount	impactCPU
8/23/2013 14:39:13.27	A524897	20,942.00	90.94	611,468.00	9,510.91
8/24/2013 15:01:26.36	A524897	21,025.00	92.15	615,608.00	9,533.95

Tuning Example #2 PPI Bounding

Tuning Example #2 - PPI Bounding

Total CPU	Total IO	TotalIMPACTCPU	TotalSpoolUsage	Daily Runs
21,555,63	40,005,096	22,205.950	726,555,136	2

```
SELECT *
FROM   PROD_V.STR_PUSH_ACT
WHERE  CRTE_BTCH_I = (
            SELECT  max(CRTE_BTCH_I)
            FROM    PROD_V.STR_PUSH_ACT
            )
  AND  UPDT_BTCH_I = (
            SELECT max(UPDT_BTCH_I)
            FROM PROD_V.STR_PUSH_ACT
            );
```

Next, we do an all-AMPs **SUM** step to aggregate from **PRODSED.STR_PUSH_ACT** by way of an all-rows scan with no residual conditions… The estimated time for this step is **1 minute and 19 seconds.**

- Combined the two max functions into 1 pass
- There is a PPI on STR_PUSH_WORK_D
- Made Business assumption that the max batch Id's would be within a 4 week period.
- Place upper and lower bounds on derived table as well as the outer table.

```
SELECT *
FROM   PROD_V.STR_PUSH_ACT
WHERE (CRTE_BTCH_I ,UPDT_BTCH_I ) IN
      (SELECT max(CRTE_BTCH_I)
                ,max(UPDT_BTCH_I)
       FROM   PROD_V.STR_PUSH_ACT
       WHERE STR_PUSH_WORK_D
             BETWEEN CURRENT_DATE - 14
             AND CURRENT_DATE + 14
       )
AND    STR_PUSH_WORK_D
             BETWEEN CURRENT_DATE - 14
             AND CURRENT_DATE + 14;
```

we do an all-AMPs **SUM** step to aggregate from a **single partition** of **PRODSED.STR_PUSH_ACT** with a… The estimated time for this step is 7.67 seconds.

The contents of **Spool 1** are sent back to the user as the result of statement 1. The total estimated time is **15.04 seconds.**

Total CPU	Total IO	TotalIMPACTCPU	TotalSpoolUsage	Daily Runs
1,315.50	6,208,080	1,520.64	726,555,136	2

Tuning Example #3 Using a Volatile Table

Tuning Example #3 - Using A Volatile Table

```
Select          *
from(
Select       B.PO_HDR_MSTR_I  ,D.CO_LOC_I
             ,E.MDSE_ITEM_I   ,A.VEND_I
             ,A1.VEND_N       ,A.PO_I
             ,A.DEPT_I        ,A.CLASS_I
             ,A.ITEM_I        ,D.CO_LOC_REF_I
             ,A.SHIP_TO_LOC_I ,A.ORIG_ORD_Q
             ,cast(A.AUD_CRTE_TS as date) as CRET_DT
             ,A.AUD_CRTE_TS
             ,RANK() OVER (PARTITION BY B.PO_HDR_MSTR_I
                   ,D.CO_LOC_I, E.MDSE_ITEM_I
                   ORDER       BY A.AUD_CRTE_TS ) RANK1
From       prodetl_v.PO_LOC_ITEM_E A
join       prod_v.PO_HDR_MSTR B
                on             B.PO_I = A.PO_I
                and            B.PO_NUM_SRC_C = A.PO_NUM_SOURCE_C
                and            B.DEPT_I = A.DEPT_I
                and            B.VEND_I= A.VEND_I
                and            B.PO_CRTE_C <> '8'
                and            B.ACTV_F = 'Y'
join       prod_v.PO_HDR C
                on             C.PO_HDR_I = B.PO_HDR_MSTR_I
                and            C.ACTV_F = 'Y'
                and            C.PO_APRO_D = cast(A.AUD_CRTE_TS as
DATE)
join       prod_v.CO_LOC D
                on             D.CO_LOC_REF_I = A.SHIP_TO_LOC_I
                and            D.ACTV_F = 'Y'
                and            D.DEL_F = 'N'
join       prod_v.MDSE_ITEM_KEY_LKUP E
                on             E.MDSE_DEPT_REF_I = A.DEPT_I
                and            E.MDSE_CLAS_REF_I = A.CLASS_I
                and            E.MDSE_ITEM_REF_I = A.ITEM_I
                and            E.ACTV_F = 'Y'
join       PROD_V.VEND A1
                on             A1.VEND_REF_I = A.VEND_I
                and            A1.ACTV_F = 'Y'
                and            A1.DEL_F = 'N' ) AA1
    where       AA1.RANK1 = 1;
```

```
/*-------original query we are recreating was all inner joins, we can
now use the much smaller VT  to drive our query.---------*/
Select  A.PO_HDR_MSTR_I   ,D.CO_LOC_I      ,E.MDSE_ITEM_I
        ,A.VEND_I         ,A1.VEND_N       ,A.PO_I
        ,A.DEPT_I         ,A.CLASS_I       ,A.ITEM_I
        ,D.CO_LOC_REF_I   ,A.SHIP_TO_LOC_I ,A.ORIG_ORD_Q
        ,A.CRET_DT        ,A.AUD_CRTE_TS
        ,RANK() OVER (PARTITION BY A.PO_HDR_MSTR_I
                , D.CO_LOC_I, E.MDSE_ITEM_I
                      ORDER    BY A.AUD_CRTE_TS ) RANK1
From    VT_PO_LOC_ITEM A
inner
  join    prod_v.CO_LOC D
    on          D.CO_LOC_REF_I = A.SHIP_TO_LOC_I
  and           D.ACTV_F = 'Y'
  and           D.DEL_F = 'N'
inner
  join    prod_v.MDSE_ITEM_KEY_LKUP E
                on              E.MDSE_DEPT_REF_I = A.DEPT_I
                and             E.MDSE_CLAS_REF_I = A.CLASS_I
                and             E.MDSE_ITEM_REF_I = A.ITEM_I
                and             E.ACTV_F = 'Y'
inner
  join    PROD_V.VEND A1
                on              A1.VEND_REF_I = A.VEND_I
  and           A1.ACTV_F = 'Y'
  and           A1.DEL_F = 'N'
qualify         RANK1 = 1;
```

Tuning Example #3 - Using A Volatile Table

```
/* ---Create initial driving table pre-joining PO tables ----*/
create volatile table VT_PO_LOC_ITEM as (
select  A.LD_BTCH_I       ,A.LD_SEQ_I            ,A.PO_I
        ,A.DEPT_I         ,A.CLASS_I             ,A.ITEM_I
        ,A.SHIP_TO_LOC_I  ,A.ORIG_ORD_Q          ,A.VEND_I
        ,cast(A.AUD_CRTE_TS as date) as CRET_DT
        ,A.AUD_CRTE_TS    ,B.PO_HDR_MSTR_I
From    prodetl_v.PO_LOC_ITEM_E A
inner
   join   prod_v.PO_HDR_MSTR B
     on   B.PO_I = A.PO_I
    and   B.PO_NUM_SRC_C = A.PO_NUM_SOURCE_C
    and   B.DEPT_I = A.DEPT_I
    and   B.VEND_I= A.VEND_I
    and   B.PO_CRTE_C <> '8'
    and   B.ACTV_F = 'Y'
inner
   join   prod_v.PO_HDR C
     on   C.PO_HDR_I = B.PO_HDR_MSTR_I
    and   C.ACTV_F = 'Y'
    and   C.PO_APRO_D = cast(A.AUD_CRTE_TS as DATE)
Group by 1,2,3,4,5,6,7,8,9,10,11,12
) with data
primary index (LD_BTCH_I,LD_SEQ_I)
on commit preserve rows;

Collect stats on VT_PO_LOC_ITEM column (LD_BTCH_I,LD_SEQ_I);
```

Tuning Example #3 - Using A Volatile Table (Cont.)

- Pre-join related PO Tables into a Volatile table, with a PI that matches PI on `MDSE_ITEM_KEY_LKUP`
- Collected Stats on Volatile table
- Use Volatile table to drive the remaining joins.

```
This forces that optimizer to join the PO tables first rather than letting the optimizer choose
what order to join in.
```

Step	Total CPU	Total IO	TotalIMPACTCPU	TotalSpoolUsage	Daily Runs
Original	29,822.26	1,087,322.00	19,337,577.98	5,424,474,112.00	1

Original Query ran in 6 hours 50 minutes

Tuned Query ran in 6 minutes **on a busy box.**

Step	Total CPU	Total IO	TotalIMPACTCPU	TotalSpoolUsage	Daily Runs
Volatile table	864.56	635,523.00	1,852.42	5,201,060,352.00	1
Select	452.21	527,111.00	34,509.31	2,974,584,832.00	1
TOTAL	1,316.13	1,162,634.00	36,361.73	8,175,645,184.00	1

Tuning Example #4 Sliding Window Join

Tuning Example #4 - Sliding Window Join

```
SELECT   SRC_SYS_CRTE_D , MTCH_FAIL_REAS_C , SUM ( MOS_ITEM_Q )
FROM     (
SELECT   mos.MTCH_FAIL_REAS_C , mos.mdse_item_i , mos.co_loc_i ,
                    MOS.MOS_ITEM_Q , PC.PCHG_ITEM_Q , str_mos_i ,
pc.pchg_det_i ,
                    PCHG_TAK_D , SRC_SYS_CRTE_D
FROM     PROD_V.STR_MOS_ITEM mos LEFT JOIN PROD_V.BRCK_MRTR_STR loc
         ON         loc.BRCK_MRTR_STR_I = mos.CO_LOC_I INNER JOIN
PROD_V.PCHG_DET pc
         ON         mos.CO_LOC_I = pc.CO_LOC_I
         AND        pc.MDSE_ITEM_I = mos.MDSE_ITEM_I
         AND        mos.MOS_ITEM_RETL_A = pc.PCHG_CURR_RETL_A
         AND        mos.SRC_SYS_CRTE_D = pc.PCHG_TAK_D
         AND        pc.PCHG_TAK_D = mos.SRC_SYS_CRTE_D
         AND        PC.PCHG_ITEM_Q = MOS.MOS_ITEM_Q
WHERE    loc.ACTV_F = 'Y'
         AND        pc.PCHG_SRC_TYPE_C = 3
         AND        mos.FIN_SYS_PRCS_D = '1900-01-01'
         AND        mos.MTCH_FAIL_REAS_C NOT IN ( 'QSBT' , 'REJ' ) ) X
GROUP    BY SRC_SYS_CRTE_D , MTCH_FAIL_REAS_C
ORDER    BY SRC_SYS_CRTE_D , MTCH_FAIL_REAS_C;
```

Tuning Example #4 - Sliding Window Join

4) We do an all-AMPs **JOIN** step from **Spool 5** (Last Use) by way of a **RowHash match** scan, which is joined to **PRODINR.PCHG_DET** by way of a **RowHash match** scan with a condition of ("((PRODINR.PCHG_DET.PCHG_SRC_TYPE_C (FLOAT, FORMAT '-9.99999999999999E-999'))= 3.00000000000000E 000) AND (NOT (PRODINR.PCHG_DET.PCHG_CURR_RETL_A IS NULL))"). **Spool 5** and **PRODINR.PCHG_DET** are joined using a sliding-window **merge join**, with a join condition of ("(BRCK_MRTR_STR_I = PRODINR.PCHG_DET.CO_LOC_I) AND ((CO_LOC_I = PRODINR.PCHG_DET.CO_LOC_I) AND ((PRODINR.PCHG_DET.MDSE_ITEM_I = MDSE_ITEM_I) AND ((MOS_ITEM_RETL_A = PRODINR.PCHG_DET.PCHG_CURR_RETL_A) AND ((SRC_SYS_CRTE_D = PRODINR.PCHG_DET.PCHG_TAK_D) AND ((PRODINR.PCHG_DET.PCHG_ITEM_Q)= (MOS_ITEM_Q))))))"). The input table **PRODINR.PCHG_DET** will not be cached in memory. The result goes into **Spool 3** (all_amps) (compressed columns allowed), which is **built locally** on the AMPs. The size of **Spool 3** is estimated with no confidence to be 16,184,145 rows (436,971,915 bytes). The estimated time for this step is 12.42 seconds.

Tuning Example #4 - Sliding Window Join

```
CREATE MULTISET TABLE PRODINR.PCHG_DET ,NO FALLBACK ,
 . . .
PRIMARY INDEX PPI_PCHG_DET ( MDSE_ITEM_I ,CO_LOC_I )
PARTITION BY RANGE_N(FIN_WK_END_D
BETWEEN DATE '2000-01-01' AND DATE '2008-12-26',
DATE '2008-12-27' AND DATE '2015-01-04' EACH INTERVAL '7' DAY );
```

```
CREATE MULTISET TABLE PRODINR.STR_MOS_ITEM ,NO FALLBACK ,
. . .
PRIMARY INDEX STR_MOS_ITEM_X2 ( CO_LOC_I ,MDSE_ITEM_I ,FIN_SYS_PRCS_D )
PARTITION BY RANGE_N(FIN_SYS_PRCS_D
BETWEEN DATE '1900-01-01' AND DATE '1900-01-01',
DATE '2009-12-01' AND DATE '2014-12-31' EACH INTERVAL '1' DAY );
```

- Solution was to create two volatile tables with the same PI and no partitioning

Tuning Example #4 - Sliding Window Join

```
Create Multiset Volatile Table VT_PCHG_DET
    (MDSE_ITEM_I INTEGER NOT NULL,
     CO_LOC_I SMALLINT NOT NULL,
     PCHG_SRC_TYPE_C CHAR(2) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
     PCHG_CURR_RETL_A DECIMAL(9,2),
     PCHG_ITEM_Q INTEGER,
     PCHG_TAK_D DATE FORMAT 'yyyy-mm-dd' NOT NULL)
Primary Index ( CO_LOC_I, MDSE_ITEM_I )
On Commit Preserve Rows;

Insert Into VT_PCHG_DET
Select MDSE_ITEM_I
     ,CO_LOC_I
     ,PCHG_SRC_TYPE_C
     ,PCHG_CURR_RETL_A
     ,PCHG_ITEM_Q
     ,PCHG_TAK_D
From PROD_V.PCHG_DET
where PCHG_SRC_TYPE_C = 3;
```

Tuning Example #4 - Sliding Window Join

```
Create Multiset Volatile Table VT_STR_MOS_ITEM
    (CO_LOC_I SMALLINT NOT NULL,
     MDSE_ITEM_I INTEGER NOT NULL,
     MOS_ITEM_Q DECIMAL(6,2),
     SRC_SYS_CRTE_D DATE FORMAT 'yyyy-mm-dd' NOT NULL,
     FIN_SYS_PRCS_D DATE FORMAT 'yyyy-mm-dd',
     mtch_fail_reas_c char(4),
     mos_item_retl_a DECIMAL(9,2))
Primary Index ( CO_LOC_I ,MDSE_ITEM_I  )
On Commit Preserve Rows;

Insert Into VT_STR_MOS_ITEM
Select CO_LOC_I
     ,MDSE_ITEM_I
     ,MOS_ITEM_Q
     ,SRC_SYS_CRTE_D
     ,FIN_SYS_PRCS_D
     ,MTCH_FAIL_REAS_C
     ,MOS_ITEM_RETL_A
From PROD_V.STR_MOS_ITEM
Where FIN_SYS_PRCS_D = '1900-01-01'
  And MTCH_FAIL_REAS_C Not In ( 'QSBT' , 'REJ' );
```

Tuning Example #4 - Sliding Window Join

```
Select mos.SRC_SYS_CRTE_D
    ,mos.MTCH_FAIL_REAS_C
    ,Sum(MOS.MOS_ITEM_Q)
From VT_STR_MOS_ITEM mos
Left Join PROD_V.BRCK_MRTR_STR loc
 On  loc.BRCK_MRTR_STR_I = mos.CO_LOC_I
 And loc.ACTV_F = 'Y'
Inner Join VT_PCHG_DET pc
 On  mos.CO_LOC_I = pc.CO_LOC_I
 And mos.MDSE_ITEM_I = pc.MDSE_ITEM_I
 And mos.MOS_ITEM_RETL_A = pc.PCHG_CURR_RETL_A
 And mos.SRC_SYS_CRTE_D = pc.PCHG_TAK_D
 And mos.MOS_ITEM_Q = pc.PCHG_ITEM_Q
Group By mos.SRC_SYS_CRTE_D, mos.MTCH_FAIL_REAS_C
Order By mos.SRC_SYS_CRTE_D, mos.MTCH_FAIL_REAS_C;
```

Status	AMPCPUTime	TotalIOCount	MaxAMPCPUTime	ImpactCPU	Step
Before	23,734	40,507,102		138,134	
After	0	8,071	0	9	Create Multiset Volatile Table VT_PCHG_DET
After	2,825	5,190,107	3	3,433	Insert Into VT_PCHG_DET
After	0	3,459	0	5	Create Multiset Volatile Table VT_STR_MOS_ITEM
After	310	350,167	0	406	Insert Into VT_STR_MOS_ITEM
After	4,875	15,877,813	38	44,214	Select mos.SRC_SYS_CRTE_D
	8,009	21,429,617		48,066	
	66.25%	47.10%		65.20%	

Top 20 Biggest Tables

DataBaseName	TableName	Size in GB	Rowcounts
PRODINV	ITEM_LOC_DAY_OH_INV	34,112	755,618,256,238
PRODRPT	MDSE_SLSTR_LITM_FCT	6,914	50,372,270,105
PRODRPT	OPER_ITEM_LOC	6,653	156,634,773,645
PRODSLS	MDSE_SLSTR_ITEM_LINE	4,055	48,992,349,648
PRODMRG	WK_ITEM_LOC_VEND_GRMRG	3,304	54,671,987,846
PRODINV	ITEM_LOC_REPLN_ATTR	3,224	77,219,478,085
PRODSLS	TRAN_ITEM_LINE_PRC_ADJ	3,137	55,527,726,430
PRODINV	ITEM_LOC_OH_AUD	2,379	27,095,542,617
PRODSHO	MDSE_ITEM_LOC	2,216	32,473,659,670
PRODSED	STR_PUSH_ACT	1,468	32,677,771,910
PRODETLWORK	RSRCH_52_WK_FCST_E	1,451	8,142,030,054
PRODRPT	DAY_MDSE_STR_SLS_CNT	1,393	33,911,788,694
PRODETLWORK	RCPT_ERR_W	1,385	9,464,618,797
PRODETLWORK	PRSMN_CALC_VAL_E	1,316	12,205,060,317
PRODETLWORK	STR_INV_E	1,268	20,754,405,577
PRODRPT	DAY_PMTN_PGM_CLR_CIRC_DIM	1,096	34,548,618,773
PRODRPT	WK_MDSE_DSPL_STR_ITEM	1,085	20,742,544,739
PRODRPT	WK_MDSE_DSPL_STR_ITEM_FIT	1,047	20,054,335,137
PRODINV	ITEM_LOC_DAY_PERSH_OH_INV	946	25,935,234,423
PRODINV	ITEM_BRCK_MRTR_STR_PRSMN	941	10,744,733,409