SQL Tuning Worksheet

I have created this document to organize/guide/remind people of essential ideas and steps in the tuning process. But this document has fought me a bit, most notably in the area of page size. I want the document to read well, especially for any pasted work a tuner puts into the document, but many times the artifacts are wide (they run off the edge of the page). Being stupid as I am, and not a super user of MS-WORD, the only thing I could figure out was to make the page size really big and use the WEBLAYOUT view. I have also attached the SQL\*Plus files used for information gathering, directly in this document so that they will never get “lost”. But this also requires that these files must be kept up-do-date by YOU, whenever I send you updated scripts. I am not sure if this is more work or less than just shipping around a zip file with the scripts. But it does look neat to have the files embedded in the doc.

This document provides the following grouped sections to help you tune and then document your success.

* Contact info (1)
* Solution Summary (2)
* Initial Information Acquisition (3)
* Details of your Analysis (4)
* Reminders of what to do (all the other sections (5-12) plus commentary where valuable)
* Helper SQL Scripts (liberally distributed throughout the document)

# Contact Information

## Your contact info

Name: Kevin Meade

Email: km133688@sbcglobal.net

Business Phone: 1.860.425.0881

 Nice, a picture that is 22 years old, shows me in a much better light (that was my favorite tie). Naturally you will replace this with your picture once you get started.

## Requestor contact info

Name:

Email:

Business phone:

Role (manager/developer/user):

Application owning the query:

# Solution Summary

Query Identity is intended to be the “name” of the query as Oracle knows it; something unique that Oracle and you can use to find the query on a system over and over. In general I tend to use the identifying data from the TOP-N report if that is how I found the query (trolling) or I ask the requestor for it or otherwise get it myself. Also, the data I use for this comes from the query I actually tune, not the query that was originally reported. Thus if tuning is done on a DEV database, then that info from that database goes in this report. Later you will get a chance to note which database first exhibited the problem. If you are tuning on a different database from the problem database, you can include additional identifying info as you deem needed.

Descriptions here should be short and sweet. Tuning details will be captured in later sections. Do not forget to mention how much better things are after your fixes. That said, this is your report, and your work, so always capture in whatever section you are working, whatever it is you want to say.

SHOWALL11G.SQL is new. Feed it data from the showtopcpu11g.sql report. That report has a helper line for it.

## Query identity



Date and Time:

Database:

DBID:

INST\_ID:

SQL\_ID:

CHILD\_NUMBER:

SQL\_HASH\_VALUE:

## What was the problem?

## What did you do to fix it?

## What is the benefit after the fix?

## Other stuff that needs saying

This section is a great place to put a review of your analysis. As I go along and figure things out, I drop a bullet item here. That way by the time the analysis is over, there is a record of all the important things that happened.

# Initial phase: information acquisition

Business INFO here is important because this ties you to the user who needs the fix. Do not be skimpy. These details will allow you to explain the problem and also the business area that is affected by the problem and this in turn will go a long way to making you look good. If you are using this worksheet to do your SQL Tuning, then you are becoming, or already are, a professional with a rare skill (SQL Tuning) and you want to present that skill in a good light. There is nothing wrong with a little trumpet blowing and these details will let you do that because they will help you give better presentations later. So do not be lazy when it comes to capturing the problem statement in your user’s words, and the business description of what the query is supposed to do. As for the rest of the info, you will not need all of it, but it is so easy to collect now with the scripts, so why not.

## Database info

Database experiencing the problem:

Database where testing will be done:

## Business info

Current wall clock runtime:

Expected wall clock runtime:

### Problem statement in user's words

### User’s definition of success

### Business description of the query

### Other Metrics

### Any other relevant notes

### TOP-N Report

@showtopcpu11g.sql

One truth of Oracle Relational Database Performance Metrics is and has always been: that 90% of the work being done on a database is being done by < 1% of the queries running on that database. This TOP-N report uses a logarithmic scale based on CPU time to bucket all SQL currently in the cursor cache, by instance, so that you can easily see the top 1% of SQL that is doing 90% of the work. It is also interesting to see the breakdown of SQL by a log scale in order to see the magnitude of this rule.



## Before tuning metadata (XML dump via script, zip it, and attach the zip file)

Dump script not ready yet.

## Basic query info

### Query text

@showsqlbyhash11g.sql



### Formatted query

<http://www.dpriver.com/pp/sqlformat.htm>

Toad

### Additional formatted query

### Your formatting notes

## Your initial review notes

### Mistakes

### Interesting observations

### Your commentary on the underlying data model

## Query details

### Starting QEP using either explain-plan or load-from-cache or GATHER\_PLAN\_STATISTICS

@showplan11g.sql

@showplan11gshort.sql

@sowplangps11g.sql

@loadlplanfromcache11g.sql

@loadplanfromhist11g.sql



### All Tables referenced by the QEP

@showplantablesunique11g.sql



### Describes for all tables referenced by the QEP

@showplandescribes11g.sql



### All Indexes for all tables referenced by the QEP

@showplanindexes11g.sql

@showplanindexcolumns11g.sql



### All Constraints for all tables referenced by the QEP

@showplanconstraints11g.sql



### NUM\_ROWS queries (estimated)

@showplannumrows11g.sql



### Count queries (actual)

@showplancountqueries11g.sql



### Plan cardinality queries (estimated)

@showplancardinality11g.sql



### Filter queries (actual)

@showplanfilterqueries11g.sql



### FRP query

@showplanfrpspreadsheetcode11g.sql



### FRP spreadsheet

Do statistics look up-to-date?

Has initial query workload been accurately estimated?

Is this a PRECISION STYLE or WAREHOUSE STYLE query?

How does this query style compare to what you saw in the QEP (PLAN\_TABLE\_OUTPUT section and PREDICATE INFORMATION section)?

How long does it take to actually scan all rows in all tables (see elapsed seconds of the FRP report) as compared to the elapsed time of your problem query?

What is your target runtime given the FRP elapsed time and the number of steps in your QEP (the Sanity Check Runtime Estimate) (valid for WAREHOUSE STYLE only)?

### Bind Variable History

@showsqlbinds11g.sql

Showsqlbindsbyhash11g.sql



### Join Hierarchy or Join Sentence

@showplandrivingtable11g.sql



### Performance Related Database Parameters

@showperfparams11g.sql



### Query Diagram (optional)

### Sanity-check runtime estimate (optional)

The Sanity Check Runtime Estimate is a stupid/silly/totally unreasonable estimate of runtime. Yet it is usually valid for a WAREHOUSE STYLE query. If you are tuning a PRECISION STYLE query then skip this. The idea is to calculate with a grossly easy formula, one possible maximum runtime of your long running query. The formula is (# of lines in the QEP / 2 \* the elapsed time (in seconds) of the FRP spreadsheet = maximum runtime of the query. Let us suppose that your four hour query has a QEP with 42 lines in it, and the FRP takes 17 seconds to run. Your calculation would be 42/2\*17=357 seconds. So we would expect the query to take somewhere around 6 minutes to do. At first this sounds really outrageous as a way to calculate runtime but in fact what we just did was define a “worst case” scenario and then put a number on it. We said that if we looked at all the data in the query for every base operation in the query plan, how long would that take? Seems dumb, and for really long plans it falls apart. But for a large percentage of QEPs, it works pretty well, and gives you an idea of when you can stop tuning.

### Data Model (optional)

### Column Statistics (as needed)

@showcolstats.sql



### Histograms (as needed)

@showhistogram.sql



### Other Useful Scripts

@showowner.sql

@showindexes.sql

@showconstraints.sql

@showmyscanrates.sql

@showmyworkareas.sql



# Details of your Analysis (your work goes here)

This is a free form area. Cut/Paste important steps and discoveries as you make them, into this section. Although other sections below may offer locations to put your work, the reality is, it is much easier just to drop everything here. Doing so will allow you to retain the workflow that occurred so that the timeline of discovery can be maintained, which in turn makes it way easier to explain how you arrived at a solution. This will be just as important to you as anyone else when you go back and try to figure out what you did. It will make more sense because reading what happened will flow better. This section can end up many pages long and that is OK. Remember to use the SQLTEXT font in order to line up column headings etc. to make code and data snippets readable.

# Use your Discovery Tools to discover the problem

SQL Tuning is mostly about discovery. You have to figure out what is wrong. The book you are reading (I hope you are reading it (Oracle SQL Performance Tuning and Optimization: It’s all about the Cardinalities)), teaches you the following discovery techniques. Use these processes/methods/ideas, and their associated scripts, to ferret out the problems in your PROBLEM QUERY.

Hints are a discovery tool.

Dynamic Sampling is a discovery tool.

Count Queries and Filter Queries are discovery tools.

Reconstruction Queries are a discovery tool.

FRP is a discovery tool.

GATHER\_PLAN\_STATISTICS is a discovery tool.

2% Rule is a discovery tool.

Query Simplification is a discovery tool.

Every section of the QEP (especially PREDICATE\_INFORMATOIN) is a discovery tool.

Data Model Meta-Data is a discovery tool.

Bad database / Good database is a discovery tool.

# Concentrate on the most common problems first

SQL Tuning is mostly about fixing a mistake. In almost all cases, your PROBLEM QUERY will be going slowly because there is a mistake of some kind; these mistakes will be one of the types noted below. These are broad categories of problem definitions for sure, but as these notes portray, you will be spending most of your time in two sections of the QEP, validating and reviewing these specific pieces of information. So get comfortable with PLAN\_OUTPUT and PREDICATE\_INFORMATION and start tuning by looking for Cardinality Errors, FILTER operations, Oracle’s mods to your query, and unusual query constructs. The notes below collectively cover the ideas of (CARDINALITY ERRORS / INEFICIENT FILTERING AND JOINING / CODING MISTAKES / OPTIMIZER LIMITATIONS).

Cardinality Errors (estimates vs. actuals) in initial plan steps of that feed the query, as seen in the PLAN\_OUTPUT section of the QEP.

Cardinality Errors (estimates vs. actuals) in Join steps, and the Access Method steps that feed these joins, as seen in the PLAN\_OUTPUT section of the QEP.

Cardinality Errors (estimates) in any sequence of steps where a large number of rows (millions?) is reduced to an amazingly small number of rows (1 or 2?).

FILTER operations in PREDICATE\_INFORMATION that suggest an indexing problem (missing index, missing columns, DUNSEL columns, column modification, implicit data type conversion, wrong index).

FILTER operations in PREDICATE\_INFORMATION that suggest a join problem (hash join across a partial key, nested loops join using a poor index, nested loops join that should have been a hash join).

FILTER operations in PREDICATE\_INFORMATION that suggest some other problem.

PREDICATE\_INFORMATION showing if and how Oracle modified your query.

Unusual or complex or stupid query constructs. Obvious candidates are: implicit datatype conversion / columns modified by functions / errors with outer-join / complex code that may not be necessary (aggregations, join-backs, etc.).

# Use Query Decomposition and Reconstruction to better understand the query and localize its performance problem

SQL Tuning is mostly a DO-ER’S art. That means you have to change things and make stuff execute to figure out what is wrong. Query Decomposition and Reconstruction is the oldest and most useful method for making that happen. It simply means that you look for ways to chop up a query into smaller pieces so that you can judge their influence on the performance of the query independently of the other parts. Any of the concepts noted below are candidates for Query Decomposition and Reconstruction, because they each offer an opportunity to cut out or chop up a query into something smaller, and thus make it easier for you to make headway toward an understanding. To tune SQL, you have to roll up your sleeves and get dirty and this is one way to do it.

Scalar sub-queries.

Function calls from SQL.

UNION/UNION All (mutually exclusive row sets).

Analytic totals.

Existential sub-queries (IN/EXISTS).

Jumping over Database Links.

Long list of Joins.

Anything that looks stupid.

Something you do not understand.

# Evaluate the query for simplification

Simplify, Simplify, Simplify! This is another stab a Query Decomposition and Reconstruction. Half of the time, your query will come in parts. Usually only one of its parts is the keeper of the problem you are trying to find. So removing pieces of a query to work on a smaller simpler query, affords you less work, since you will be able to either eliminate parts of the query early on as not having the problem you are looking for, or you will isolate the problem to one of the pieces early on. Either way, you have moved way forward since you now have something much smaller to work with, which leads in particular, to better use of your mental energy, and shorter runtimes for your tests. So get to it, and revisit Query Decomposition and Reconstruction again.

# Discovery Phase: The Easy Part is over, now you have to think: (always capture important pieces of your work as you go so you can explain your success to others)

Notice the heavy emphasis here on first making sure that you have a feeling for how the query should behave (its style), and for looking at the query plan as being composed of three major grouping of steps (initial data feeds, joins, other). This is not a coincidence. The style guides you to think a specific way about what should be happening, and looking at the QEP in isolated groups again lets you focus on central ideas. Almost all SQL Tuning problems will manifest in either Initial Plan Steps, or Join Steps, so evaluating the QEP with this emphasis and in this order, is highly useful.

## Classify the query style as PRECISION STYLE or WAREHOUSE STYLE using the FRP spreadsheet (or other methods)

What is the basic query style?

### Your work goes here

## Validate if the query got off to a good start (initial data acquisition steps of the QEP)

Are statistics up-to-date (FRP Spreadsheet)?

Do estimates match actuals (FRP Spreadsheet) (GATHER\_PLAN\_STATISTICS) (use count/filter/reconstruction queries if necessary)?

Does driving table look reasonable for the query style?

Does join order look reasonable?

Do access methods look reasonable for the query style (2% Rule)?

If access methods use indexes, are these indexes being used correctly?

### Initial Data Acquisition Steps of the QEP

### Your work goes here

### Correct any problems and revalidate

## Validate joins and the access methods that feed them

Do estimates match actuals (GATHER\_PLAN\_STATISTICS) (use count/filter/reconstruction queries if necessary)?

Do access methods look reasonable for the joins they feed?

Do join methods look reasonable for the query style (2% Rule)?

Does any join suffer from a bad cardinality estimate?

Is Hash Join using only a partial key because of inequality predicates?

Is Hash Join OPTIMAL, ONE-PASS, or MULTI-PASS?

Is HASH JOIN using TEMP STORAGE?

Is Nested Loops Join inner table using an index with missing columns?

Is Nested Loops Join inner table using an index with DUNSEL columns?

Is Nested Loops Join inner table using the wrong index?

### Join Steps of the QEP

### Your work goes here

### Correct any problems and revalidate

## Validate "everything else" plan steps

Do estimates match actuals (use count/filter/reconstruction queries if necessary)?

### “Everything Else” Steps of the QEP

### Your work goes here

### Correct any problems and revalidate

## Validate database parameters (not likely the problem but do it anyway)

If there are multiple environments involved, are there differences between environments?

Do the performance related parameters suggest any ideas?

### Database Parameters

### Your work goes here

### Correct any problems and revalidate

# Solution Testing Phase: Evaluate your position

In the end, to fix whatever problem you have found, you will have to change something. The items noted here are the things you will be changing. Testing means just that, testing. You will have to do lots of testing in two ways. First you will need to show that your fix provides a massive performance boost and thus solves the performance problem you have. But second, you may have considerable testing you need to do to demonstrate that you nave not changed the answer being returned by whatever it is you are tuning. This will always require the copying of data, building of before/after result sets for comparison, etc. You must do this testing and show it.

Consider statistics changes (staleness, skew, dependence, defaulting, out-of-bonds, bloat, confusing predicates: (new stats, extended stats, histograms, dynamic sampling))

Consider data model changes (indexes/constraints/data types/partitioning)

Consider query refactoring (better algorithms, simplification, lazy bug fixes)

Consider database parameter changes

Consider advanced features (parallel query, partitioning, partition wise hash join, manual memory management)

Consider solution redesign (row-by-row vs. set based, pre-summarized data, work redistribution)

Consider not a SQL Tuning problem

### Your work goes here

# Rinse and Repeat

Keep thinking till you get an answer.

# Capture your final solution

## Description of your solution (also update summary section)

## After metadata (XML dump, zip it and attach the zip file)