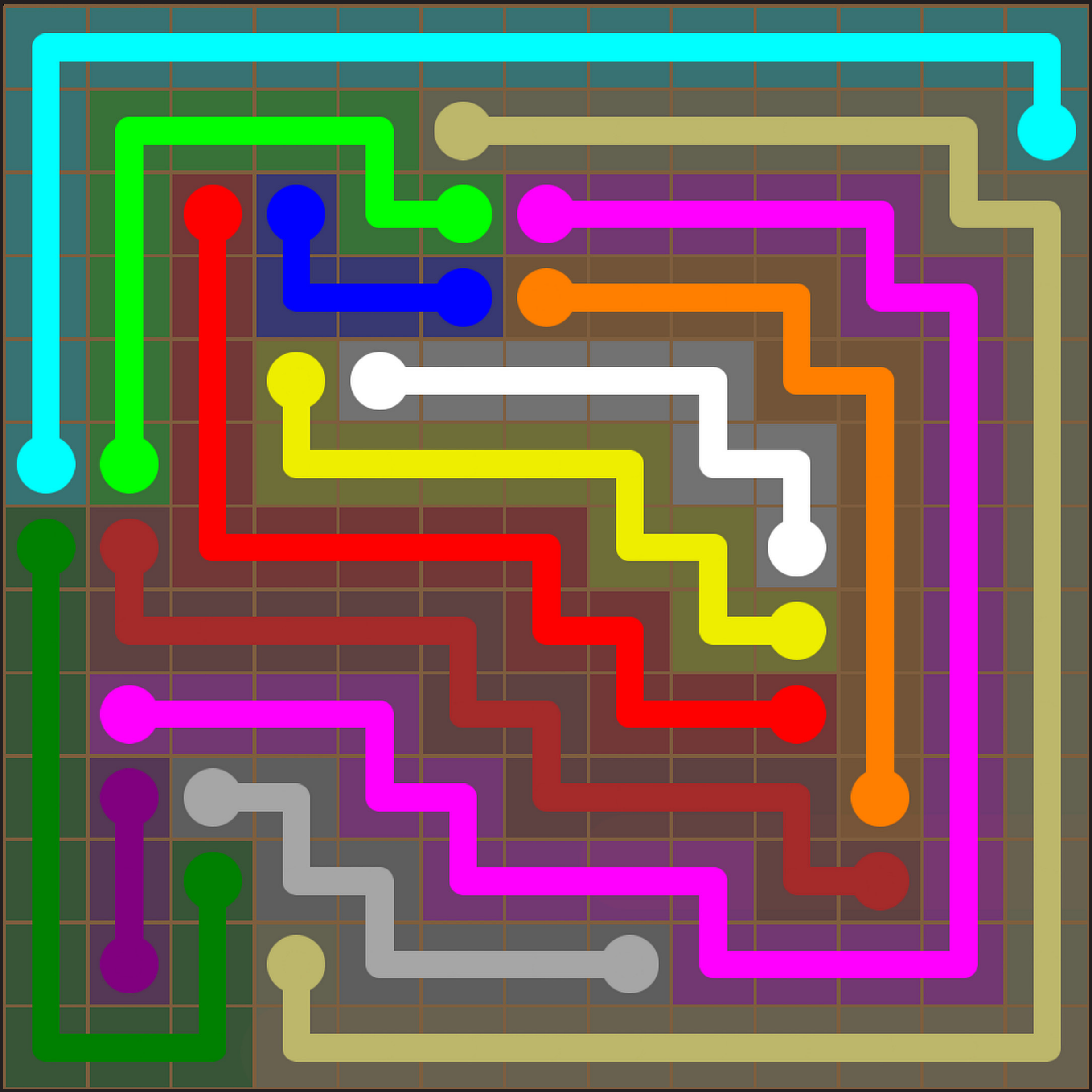
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 29, No. 2 · MAY 2015 $15



Women in Technology

With Gwen Shapira.

See page 4.

Tuning Oracle SQL with SQLTXPLAIN

Book review.

See page 6.

Fastest is Not Always Shortest

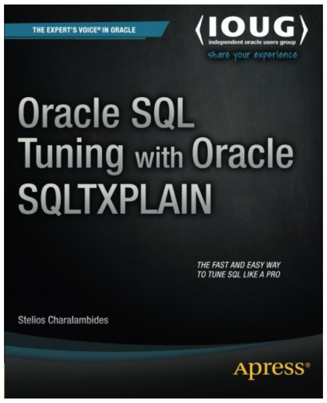
Unconventional wisdom.

See page 19.

Much more inside . . .

Oracle SQL Tuning with SQLTXPLAIN

A Book Review by Brian Hitchcock


THE EXPERT'S VOICE® IN ORACLE
IOUG
independent oracle users group
share your experience
Oracle SQL Tuning with Oracle SQLTXPLAIN
THE FAST AND EASY WAY TO TUNE SQL LIKE A PRO
Stelios Charalambides
apress®

Details

Author: Stelios Charalambides

ISBN: 978-1-4302-4809-5

Pages: 317

Year of Publication: 2013

Edition: 1

List Price: $44.99

Publisher: Apress

Summary

Overall review: Very good; a clearly written description of what SQLTXPLAIN does and how to use it effectively.

Target audience: Anyone who needs to tune SQL in an Oracle environment.

Would you recommend to others?: Yes.

Who will get the most from this book?: Anyone who is working with individual SQL performance issues.

Is this book platform specific?: No.

Why did I obtain this book?: NoCOUG asked me to review this book.

Overall Review

SQLTXPLAIN is a powerful utility for researching and resolving Oracle SQL performance issues, and this book explains how to use it very well.

Before we go any further, I want to discuss my experience with Oracle SQL tuning in general. I have worked in various IT environments for over 20 years, and I have never had a test environment that was dedicated to testing SQL performance, that was a duplicate of production, or that had any way of simulating the production load. Further, in my experience management was never willing to invest the considerable time and money needed to really track down the issue and resolve it. I don't say this to be mean-spirited or to imply that SQL tuning isn't important. I say this to make sure you understand that when I read this book, I knew I wouldn't be tuning SQL anytime soon and I didn't know what it would be like to have test environments available to me.

Book quality: With the advent of print-on-demand books and the demise of what we now call "traditional" publishing, I've noticed a substantial decline in the quality of the printed books I review. In this book the figures are fuzzy to the point that I can't read the column headings. I found 25 typos in the text, and I'm not a professional copyeditor and wasn't even looking for typos.

Author's Introduction

The author explains that this is a practical guide to using the SQLTXPLAIN tool, how the book came to be written, and who should and should not buy this book. If you are looking for tuning theory, you are advised to look elsewhere. At the end of this section, we are told that tuning problems are the most complex issues. This book is focused on SQL statement tuning; from my experience, the real performance issues that I have seen recently involve over a dozen servers supporting a database, multiple WebLogic Servers, Web Servers, and associated ID management systems. I believe this level of complexity is more typical of modern commercial environments, and I think the issues we see in these systems are the most complex. The issues of tuning only a single SQL statement in a single database can be complicated, but the context is less involved.

Chapter 1—Introduction to SQLTXPLAIN

Here we learn what SQLTXPLAIN is and its history. (Note that SQLTXPLAIN will be referred to as "SQLT" from here on.) I had never heard of SQLT before, and the author confirms that I am not alone. We also see that SQLT will not fix your problem SQL; it will help you see what is needed to fix it. I noticed that the discussion starts out assuming you have the one and only problem SQL already identified before you start looking at SQLT. How you decide which single SQL statement is worth tuning is not covered. I wonder if the author uses AWR reports or the ASH utility. The process of getting a copy and installing SQLT comes next. Note that you have to download and install the version of SQLT for your specific version of the Oracle database. You also need to have a license for the tuning and diagnostics packs to make full use of SQLT.

The steps needed to run your first SQLT report are shown. There are two different reports available: SQLXTRACT and SQLTEXECUTE. You can probably guess that SQLXTRACT reports on data found in the database for SQL that has already executed, while SQLTEXECUTE generates the data needed by actually executing the SQL in question. I think being able to execute the SQL is problematic in most production systems, especially if the SQL is altering data.

Starting with screenshots of the SQLXTRACT report that has been generated, we learn about the structure of the report, cardinality and selectivity, and what cost means to the optimizer. The execution plan section of the report is illustrated in detail, and join methods are discussed.

Chapter 2—The Cost-Based Optimizer Environment

In this chapter, and in most of this book, the way the CBO works is discussed as we learn about using SQLT to diagnose SQL performance issues. The process is a gentle introduction to this subject and would be easy for new DBAs to follow. We learn that the CBO generally makes good decisions if it is given good information.

I found the discussion of system statistics interesting because I don't think I've focused on this before. There are sections for CBO parameters, hints, and the history of the performance of the execution plan. This was, like many things in this book, something I had not seen before. I can see how this information would be very useful when performance suddenly goes bad. The discussion of column statistics, cardinality, and out-of-range values is quite detailed.

The next section presents a case study where a SQL has suddenly slowed down, and SQLT is used to find when the execution plan got worse, along with the index that was added around the same time. It is assumed that you can execute the problem SQL repeatedly. I can see this if the SQL is only doing select, but anything that changes data would require a refresh of the data between each execution. This also assumes you have a test system that accurately models production, which I have not seen in my career.

Chapter 3—How Object Statistics Can Make Your Execution Plan Wrong

The message here is simple: generating good statistics on the objects in the database is critical. The discussion covers the many ways that good statistics are not gathered, including sample size issues, histograms being used improperly, and many others. It is recommended that a pre-production test system be used to validate the statistics-gathering process before going to production. It's a good, logical idea that I have never seen in practice. I must work at bad places; I just don't see these good ideas actually happening! Another case study is given, this time involving a table that gets truncated every night.

Chapter 4—How Skewness Can Make Your Execution Times Variable

First we see what skewness is and what it means. I had heard of this in the past, but I had not seen such a clear explanation of how to quantify it. Basically, it means that some predicates will match largely different numbers of rows in a table. The advice is to understand your data, which I agree with, but I support hundreds of databases and I'm not being paid to look at the data in any one database for any length of time. A specific process to measure skewness is shown and how much is too much is clearly defined, which is good. The author gives us something we can really measure in our own data.

Next we see how skewness can upset the optimizer, followed by a lengthy discussion of histograms, their various types, and when to use them. Bind variables and their effects on the execution plan are presented, followed by cursor sharing. The detail given about the various values available for the cursor-sharing parameter was new to me, and I have a better understanding of this now.

Another case study shows how execution time can change due to poor use of histograms.

Chapter 5—Troubleshooting Query Transformations

Query transformations are described and illustrated with several examples. Somehow we find ourselves in Disneyworld; the trip to get there is an analogy for query optimization and Mickey comes in for some criticism! Oh snap! I'd advise the author to stay away from Mr. Mouse and his highly optimized team of lawyers!

Next we learn about the 10053 trace file, which documents how the optimizer decided on the execution plan. It turns out that SQLT generates this trace file automatically.

Specific query transformations that are covered include subquery unnesting (sounds awkward), complex view merging (check your mirrors), and join predicate pushdown, (which doesn't sound real to me but is).

Specific hidden database parameters are listed, as well as specific hints that affect how the CBO uses query transformation. This ties in to sections of the 10053 trace file, where we can see what parameters were in place, both normal and hidden. A trace file for a specific SQL statement is shown illustrating where we can find details of which, if any, query transformations were used.

The proper way to use hints is covered, followed by an example where the join method is changed. It was interesting to see an example where conflicting hints were specified and what the optimizer did in that case.

Finally, we see sections of a 10053 trace file used to show the actual cost computations that the optimizer performed as it examined various options.

Chapter 6—Forcing Execution Plans Through Profiles

The topic of SQL profiles was new to me. If a SQL statement has been running well in the past and suddenly runs poorly, you can use SQLT to generate a script that will create the SQL profile from when the execution was good. This script can then be used to apply the SQL profile to the SQL in the present to force it to run well again. We are then told that the SQL profile doesn't actually force anything but is more like a "super hint." This is faster than gathering statistics and looking for other causes of the performance issue. The SQL profile allows you to freeze the execution plan of a specific SQL statement. Note that once you do this, the optimizer won't change the execution plan and, over time, the profile may not be the best plan.

An example of such a script is given, and we see that the SQL profile tells the optimizer how to handle separate "query block names." These are generated by the optimizer as it transforms and examines the original SQL.

We are shown how to use one of the many parts of the SQLT tool to generate the SQL profile. The author tells us that before doing this, we must get Oracle Support to approve what we are doing. I assume this means that using a SQL profile can be dangerous, but no explanation is given.

This script is executed, which generates the script to apply the SQL profile. This script is shown in detail, with specific sections highlighted so we can see where the profile is being applied. This script is executed to actually apply the SQL profile, and the output of this is also shown. An example is given to show the execution plan, which includes a message confirming the use of a SQL profile.

The chapter ends with an example of how to generate the SQL profile on one database and apply it to a remote database.

Remember: don't use a SQL profile without talking to Oracle Support first!

Chapter 7—Adaptive Cursor Sharing

This chapter covers the many details of adaptive cursor sharing. The author describes this topic as one of the most confusing and misunderstood areas of the optimizer. I have heard of this before, but I've never understood what it is, how it works, and why it is important. This was introduced in 11g to deal with changing bind variables. The need for bind variables is covered, and the cursor_sharing parameter is shown. Bind peeking is explained as well as bind sensitive and bind aware cursors. An example is shown where SQL is set up to see how all of this works when a table has skewed data. The following SQLT report shows what the optimizer did for table data that was rare and common. I learned a lot from this discussion, I had not seen this level of detail before.

Chapter 8—Dynamic Sampling and Cardinality Feedback

This chapter deals with something I've always wondered about: what happens when the optimizer finds a table with statistics that are clearly bad or nonexistent? Dynamic sampling is explained as well as cardinality feedback, which I had not heard of before. We see how to set up dynamic sampling, and sections of a 10053 trace file show us how the optimizer uses this feature. Cardinality feedback is activated using a hidden parameter that causes the optimizer to store information about each step of the execution plan and compare this data over multiple executions. If these data show a problem, the optimizer will sample the table data. It is noted that dynamic sampling is used as a last resort. If it is happening, you need to gather better statistics! The example SQL discussed runs well on one system and less so on another. The SQLT report is used to find the bad statistics on the system that has the poor performance.

Chapter 9—Using SQLTXPLAIN with Data Guard Physical Standby Databases

I hadn't thought about it, but since SQLT requires installation in your local database, how would you use it to look at SQL tuning issues that might come up in your Data Guard physical standby? If you are running reporting on the standby database, you may run into SQL tuning issues you don't see on the primary OLTP database. Data Guard is described briefly, and we are reminded that we must often practice whatever disaster recovery process we have. And I am reminded that I have never worked anywhere that did this. The few times there was a process in place, I never saw it used because no one was confident it would work. Another piece of the SQLT toolbox is described, SQLTXTRSBY, which runs in your local database but goes out to the Data Guard physical standby database to gather the data it needs.

It is no surprise that there are limitations on this process, and these are described. Not all features of SQLT are available when looking at SQL running on the standby. How to use XTRSBY is shown and a sample output is discussed.

Another piece of the SQLT toolkit is described, roxtract, which is used on the standby to gather some of the data that XTRSBY can't.

Chapter 10—Comparing Execution Plans

I have never had this situation, but if you have one database where a SQL statement is running well and another database where it isn't, you can compare the execution plans of both to fix the bad one. I wonder how often this comes up in practice, but I realize my experiences are just that: only what I've seen. The SQLTCOMPARE tool is used to generate reports on both databases, bring the output together, and generate a report showing how the execution plans are different. This tool can be used to look at SQL execution across different platforms. I can see this being useful, for example, in a hardware migration scenario. A practical example is shown, following the many necessary steps.

Chapter 11—Building Good Test Cases

Test cases, we are told, are for situations where the solution to the tuning problem is not obvious. Assuming all the usual causes of poor SQL performance have been eliminated, using test cases may be the next thing to try. A test case allows for testing all sorts of things away from the main environment. Building the test case in another environment is complicated, and SQLT can help. The tools that build the test case don't usually copy the data (that could make the test case too large), but they can. Test cases can be run on a different platform from the source. Note that the scripts that create a test case make lots of changes to the system where they run, so you must only run them in an environment where you can quickly rebuild everything. The scripts collect all the metadata and statistics needed to reproduce the issue.

Building a test case is described through a detailed example with lots of output shown. I found it interesting that the test case doesn't need the table data because it has all the statistics from the tables in the source system. The test case will result in the same execution plan in the test system even though there is no data.

Chapter 12—Using XPLORE to Investigate Unexpected Plan Changes

The XPLORE tool is used to measure the impact of changing database parameters. The tool uses a list of the database parameters and changes one at a time to see how the execution plan is affected. The tool was originally designed to do this brute-force analysis to look for bugs in the optimizer when a database upgrade was being tested. How to use this tool is shown and limitations are discussed. XPLORE won't tell you about bad statistics, for example. The fix control facility is described, which allows you to turn specific bug fixes on and off. I had not heard of any of this before. Clearly, this should only be done on a test system.

I also had not heard of the SQL Monitoring Report. This feature is turned on by adding a hint to your SQL, and statistics are captured as the SQL executes. A detailed example is shown where XPLORE is set up and executed, and the SQL monitor reports are generated.

Chapter 13—Trace Files, TRCANLZR and Modifying SQLT Behavior

SQLT has several tools to help us look at trace files. The 10046 trace file, what it does, and how to use it are described. TKPROF, which is not part of SQLT, is described, followed by TRCASPLIT, the tool in SQLT that processes 10046 and 10053 trace files and generates a more useful output. The TRCANLZR is used to generate one set of aggregate information from multiple trace files, for example, when SQL is processed in parallel.

Chapter 14—Running a Health Check

SQLT, as useful as it is, must be installed in the local database. In situations where you can't install SQLT, you can use a separate tool called SQL Health Check (SQLHC). This is a free download from Oracle. It is a set of scripts that do not install anything in the local database and generate a subset of the information you get from SQLT. Running SQLHC is shown, as is the report that is generated. Several of the individual scripts are described in detail.

Chapter 15—The Final Word

This final chapter opens with a summary of the entire book. Tuning methodology is discussed in general to show how SQLT fits into the overall process. SQLT is described as the best tuning utility for focusing on one SQL statement because it gives you the overall picture in a format that is easy to navigate.

Appendix A—Installing SQLTXPLAIN

This appendix shows, in detail, the steps to install SQLTXPLAIN. The screen output you will see during the installation is shown, along with notes about possible issues. For example, the tool requires about 2 MB of tablespace and requires a local database connection. The possible licensing levels are covered, as well as how to install the tool remotely and how to automate the installation, i.e., a non-interactive or silent install. Finally, the process to uninstall is explained.

Appendix B—The CBO Parameters (11.2.0.1)

The XPLORE method of SQLTXPLAIN reports the effects of changing CBO parameters. To support this, SQLTXPLAIN contains a list of all the CBO parameters it uses. This appendix lists all of these parameters and includes hidden parameters. There are explicit warnings that we should not use any of the hidden parameters without first consulting with Oracle Support. Selected hidden parameters are described in detail, followed by the complete list of parameters that SQLTXPLAIN can use, for version 11.2.0.1.

Appendix C—Tool Configuration Parameters

This appendix covers all of the available configuration parameters for SQLTXPLAIN. Each parameter is described as well as the requirements to make use of the parameter and dependencies between parameters.

Conclusion

This book is an excellent introduction to a set of SQL tuning utilities that I had not heard of before. If you are tuning individual SQL statements and you have a test database, the various tools that make up SQLT should be very useful to you.

Postscript—Questions I have for the author

While reading the book, the following questions came to mind. They are not answered in the book so I'm hoping that the author will read them and take the time to respond.

- Tuning takes time. How do we know in advance that the time we spend tuning will be worthwhile? If we spend 10 hours tuning, how do we know that we will save 10 hours of database performance over the next week or month or quarter?
- How do we quickly determine that a specific tuning problem is worth the time we think we will need to fix it?
- How long do we pursue a tuning situation before we decide it isn't worth the time we think we will save with the improved SQL?
- My experience is only that: what I have experienced. In your experience, do most DBAs work in environments where they have duplicate or test databases where they can test SQL with the same data and the same workload as the production system?
- In my many years of DBA work, I have never seen a disaster recovery setup that was actually used. I have worked in several places that had what was called a disaster recovery system, but I never saw it actually used. In your experience, how many organizations actually switch between primary and disaster recovery systems on a regular basis?
- The group I work in now supports multiple customers with hundreds of databases. We have no idea what the data looks like in any one customer's database. To tune effectively, we need to "know the data" in the database. Can I effectively tune SQL in this environment?
- It has been many years since I have seen a customer's system where the performance was limited by the database. The database is almost always much faster than all the middle tiers that are doing application processing between the database and the end user. How relevant is SQL tuning for modern, multi-tier application environments?
- SQL tuning focuses on one problem SQL statement. How realistic is this? How often do complex systems have one and only one SQL that is causing all the performance issues?
- Here's a specific example from my recent experience: One of my coworkers received a customer complaint about slow performance. They reviewed the system and decided that several new indexes would help the slowest SQL run faster. They spent a week preparing new indexes and got the customer to approve the downtime to make the needed changes. But after the new indexes were implemented, performance was worse. The SQL that was taking the most resources had changed during the week spent tuning. The problem SQL the week before was not the problem SQL a week later. The customer was not happy. How could this have been handled better? ▲

Brian Hitchcock worked for Sun Microsystems for 15 years supporting Oracle databases and Oracle Applications. Since Oracle acquired Sun, he has been with Oracle supporting the On Demand refresh group and, most recently, the Federal On Demand DBA group. All of his book reviews and presentations—and his contact information—are available at **www.brianhitchcock.net**. *The statements and opinions expressed here are the author's and do not necessarily represent those of Oracle Corporation.*