Advanced Oracle SQL Tuning v3.0

by Tanel Poder | https://blog.tanelpoder.com/seminar

Training overview

This training session is entirely about making Oracle SQL execution run faster and more efficiently, understanding the root causes of SQL performance problems and Cost Based Optimizer misbehavior.

Instead of looking into *increasing* system utilization (by using more buffer cache or more parallelism) as a first "tuning" choice, this class is mostly about *decreasing* resource usage of SQL execution plans by improving their shape and available access paths.

You will learn the full range of techniques for SQL optimization starting from optimal data access and processing strategy all the way to advanced topics such as comprehensive SQL hinting and adjusting optimizer's row-count estimates.

Some SQL performance problems cannot be cured by simply adjusting optimizer behavior or improving stats, so a significant part of this training also concentrates on good database design, indexing practices and writing high performance SQL statements.

Finally, this class is not just a limited list of "tips and tricks", it will go deep into SQL execution fundamentals and data flow internals. This gives you the knowledge and tools for systematic tuning of SQL with any complexity.

Training Duration and Details

Duration:	**5 days** (10 x 4h online training spread over 2 weeks)
Location:	Online webinar (8am-12pm PST each day)
Audience:	Senior DBAs, Senior Developers, Database Architects & Designers
Skill level:	Intermediate to Advanced
Seminar format:	Tanel presents online, shows demos, answers questions
Q&A:	Attendees can ask questions any time by typing them into chat
Labs:	Tanel shows demos based on real life examples. Attendees can download the demo schemas & scripts for their own use

Objectives

- Be able to quickly diagnose and fix SQL performance problems
- Gain deep understanding of SQL plan execution internals
- Know what to change to improve SQL performance without trial and error
- Be able to control SQL execution plans and guarantee their stability
- Be able to systematically index tables for data access performance
- Help CBO to find good plans, to keep the manual tuning to minimum

After this class you won't need to memorize every single SQL performance problem or optimizer issue out there, instead you will be able to systematically work out the problem root causes yourself!

Non-objectives

- This is not a beginner "How to write SQL" class, although we will talk about *re*writing SQL for optimization purposes
- This is not a PL/SQL writing or PL/SQL code optimization class, although we will cover the benefits of bind variables and array processing that include PL/SQL examples too
- This class does not go deep into Exadata-only features, but everything taught here applies to Exadata too (including full table scans and partition pruning topics)
- This is not a database troubleshooting & instance tuning class, visit https://blog.tanelpoder.com/seminar for other classes

Recommended Prerequisites

- You'll need to be familiar with Oracle Database concepts and SQL development
- You'll need to generally know what the following Oracle keywords mean:
  shared pool, SQL parsing, cursor, buffer cache, table partition, bind variable, logical & physical IO, sort area & hash area, PGA

Deep Dive Sessions

Quick Start & Introduction

- Getting execution plans and related performance data
- Identifying where in the plan most of the work is done
- Identifying inefficiencies & Cost Based Optimizer mistakes at a glance

Deep Dive 1: Understanding SQL Plan Execution

- Deep understanding of how exactly Oracle executes SQL statements, SELECTs & DMLs
- Understand the fundamental factors affecting query execution performance
- How to use a *systematic approach* for tuning SQL of any complexity

Deep Dive 2: Accessing Data Efficiently

- *Systematic* indexing – which index type to use, when and how
- How to decide which columns and in which order to index without trial and error
- Understanding index-based data retrieval overhead
- Understanding index maintenance overhead
- When does a full table scan perform better than indexes?
- Static & dynamic partition pruning

Deep Dive 3: Processing Data Efficiently

- Optimizing table joining performance
- Monitoring SQL work-area (PGA) usage

- Keeping SQL work-areas small
- Optimizing GROUP BY and sorting performance
- Approximate data processing
- Optimizing Analytic functions (window functions)

Deep Dive 4: Controlling SQL Execution Plans

- How to make a SQL execution plan do exactly what you want
- How to achieve SQL plan and performance stability
- Controlling Adaptive features and dynamic sampling
- Understanding why SQL execution plans unexpectedly change and what to do about it
- Understanding why SQL performance can suddenly drop, when "nothing" has changed – and what to about it
- How to optimize SQL when you cannot change the application code?

Deep Dive 5: Helping the Cost Based Optimizer to Find Good Plans

- Understand what exactly the Optimizer Cost is, which input data is used for cost calculations and where does this number matter
- How to configure Oracle, statistics and CBO to keep the need for manual SQL tuning to the minimum
- How to troubleshoot CBO cost and row count misestimation
- Dealing with optimizer bugs

Is this seminar for me? Will I be able to understand it?

I often get these questions. My answer is that if you like my blog and YouTube videos, you'll like my training too! So, just browse around in the archives and see for yourself!

- https://blog.tanelpoder.com

I have a YouTube channel too – you can check out some of my informal hacking sessions there:

- https://youtube.com/TanelPoder

If you have further questions about this class, just send us an email!

- seminars@poderc.com

Table of Contents, Details & Keywords

Here are the keywords and the structure of what I plan to cover. The final contents are subject to change, I'll probably add new and remove old stuff as I update the slides with latest thinking, features and techniques.

Introduction & Quick Start
On Day 1 we begin with a "Quick Start" introduction and a high-level overview to immediately give you basic tools and techniques for understanding the performance of any SQL statement. We will look into fundamentals of execution plans, reading SQL Monitoring reports and low-hanging fruits to look at when diagnosing misbehaving SQL statements.

Deep Dive 1: Understanding Execution Plans
From Day 2 onwards we will drill down deep into low-level fundamentals, explaining how *exactly* the Oracle SQL plan execution works, what are the main factors affecting query performance and how to systematically diagnose and fix SQL performance problems.

In this part you will learn how exactly Oracle executes SQL execution plans. Instead of describing on PowerPoint sides how things might "logically" happen in Oracle, we will take a deep dive into the exact sequence of how Oracle physically executes the row sources in any execution plan and how does the data flow during plan execution.

You will learn how access path operators fetch rows from database segments and how the rows are passed through the execution plan tree throughout the lifecycle of SQL execution. You will also learn how row joining, filtering and comparison works, such as nested loop/hash joins and also the frequently seen FILTER operation.

The understanding of how SQL plan execution physically works will become a strong and required foundation for any systematic SQL tuning and troubleshooting task.

Understanding SQL Plan Execution and Data Flow

- A detailed look into how SQL plan execution works in Oracle
  - Walking the execution plan tree
  - Understanding row source operators
  - Buffering vs. cascading row sources
- Getting execution plans from the right source
- Reading execution plans the right way
- Identifying where the response time is spent
- Profiling SQL plan execution
  - Real time SQL monitoring and V$SQL_MONITOR

- Active Session History execution plan line-level info
- V$SQL_PLAN_STATISTICS_ALL
- SQL Trace STAT lines

Predicates and data filtering
- Access vs. filter vs. storage predicates
- INTERNAL_FUNCTION
- Pushed predicates

Understanding how the data joining and correlated lookups work
- Nested Loop Joins
- Hash Joins
- Merge Joins, Band Joins
- Correlated subqueries and FILTER operation

Understanding the fundamental factors affecting query performance
- Row counts fetched from a row source (cardinality)
- Block visits and revisits
- Physical vs. Logical IO and locality of data
- Consistent reads & DML: Amount of data changed (redo/undo size)
- Concurrency: Lock and latch contention
- PX: Parallel execution data fetch and distribution mechanism
- Callouts from SQL execution plan (PL/SQL function calls from SQL)

Understanding why SQL performance changes when the execution plan remains the same
- I/O performance changes and CPU availability
- An adaptive plan switched to a different join method
- Amount of work area memory allowed to use by work area manager
- Hash multipass join decisions
- Dynamic hash input swapping
- Number of parallel slaves currently available
- Runtime partition pruning decisions
- Runtime prefetching decisions
  - Nested loop prefetching
  - Index multiblock read prefetching
  - Direct path read segment scan prefetching

Deep Dive 2: Accessing Data Efficiently

In this part we will go deep into efficient ways of retrieving data from the database (accessing rows using table, index access paths and table scans). We will look into how to systematically detect inefficient data access even if nobody is complaining (yet) about performance. A large majority of this section focuses on indexes, how they work, how to use them, how to avoid using them. Additionally, we'll touch the topic of full table scans and partition pruning.

After completing this part, you will be able to systematically work out which indexed columns and in which order would give the best results for a workload – there's no need for repetitive trial and error!

Mantras of efficient SQL execution

- Don't *request* too much work!
  - Let the database know about your intentions
  - Request only the columns you need
  - Request only the rows you need
  - Request only the detail you need
  - Help Oracle to *know* the data

- Don't *do* too much work! (Simple, huh? ☺)
  - Retrieve data efficiently – reduce number of IO-s per row retrieved by access path
  - Filter data early – eliminate non-matching rows close to the access path
  - Process retrieved data efficiently – avoid heavy loops and iterations

Achieving efficient data access

- Understanding the data retrieval overhead of different access paths
- Identifying significant "throw-away" of retrieved data
- Using the right access paths for retrieving data
- Filtering data early

Systematic indexing

- How to know in advance which indexes would help with data retrieval performance
- Calculating the best column order for multicolumn indexes
- Understanding lookup overhead of indexes
- Understanding why the index clustering factor matters so much
- Understanding index maintenance overhead

Indexing strategy for high-concurrency updates and insert distribution of "right hand" indexes (and RAC)

- Reverse-key indexes
- Index column prefixing with instance ID or a deterministic function
- Scalable Sequences for primary keys

Best practices for bitmap indexing

- Avoiding heavy DML on bitmap indexes
- Achieving efficient bitmap index combining
- Achieving star transformation in star schemas

Deep Dive 3: Processing Data Efficiently
After the data retrieval steps at access path functions in the execution plan return records from the data blocks, the rows need to be processed by subsequent steps in the execution plan.

Millions of retrieved records can be collapsed to smaller human-readable result sets by joins and aggregations – and further enhanced by various analytic functions and sorting operations. These operations often require temporary memory work areas stored in Oracle process private memory and sometimes become performance bottlenecks, when the amount of data to be processed is large.

Optimizing table joining performance

- Understand the fundamental difference between lookup joins and scan & filter joins
- Reducing join memory usage
- Using partition-wise joins

Monitoring SQL work-area (PGA) usage

- Detecting inefficient multi-pass work-area operations
- Estimating PGA memory use
- V$SQL_WORKAREA & V$SQL_WORKAREA_ACTIVE
- SQL Monitoring reports

Keeping SQL work-areas small

- Optimizing GROUP BY performance
- Optimizing sorting performance
- Approximate data processing

Optimizing Analytic functions

- Reducing cardinality in analytic functions
- Narrowing sorting scope for analytic functions
- Approximate window processing

Deep Dive 4: Controlling SQL Execution Plans
In this part you will learn how to make a SQL execution plan do exactly what you want it to do. We will look into the full range of techniques such as query rewriting, full hinting, stored outlines, SQL profiles, adjusting row count estimates and passing optimizer parameters into individual statements.

We will start from identifying important prerequisites to any systematic SQL tuning, such as understanding the amount and “shape” of data accessed and what is the SQL statement supposed to do. This allows you to understand what the ideal execution order and hierarchy would be – which you can then achieve with the techniques covered next in this section.

As no SQL tuning technique is appropriate in all situations, we will also look into how to pick the right technique for fixing the right problem and common SQL optimization pitfalls.

Three main properties of an execution plan - data retrieval

- Query block layout
- Join order (execution order and hierarchy)
- Join methods
- Access paths
- Filter/aggregation placement

Controlling execution plans

- Picking the right join order
- Picking the right join methods
- Picking the right access paths

Understanding query transformations

- Heuristic query transformations
- Cost based query transformations (CBQT)
- Simple view merging
- Complex view merging
- Subquery unnesting, antijoin, semijoin conversions
  - IN, NOT IN, EXISTS, NOT EXISTS
- Join elimination transformations
- Predicate move-around
- Others…

Controlling work area operations and reducing memory usage

- Sorting, grouping and other work area operations

Making the SQL execution plan do exactly what you want – and stick to it

- Hinting the right way
- Hinting pitfalls and what to avoid
- Using SQL Plan Baselines
- Injecting hints to existing queries without changing application code
- Strategy and techniques for achieving plan stability

Deep Dive 5: Helping Cost Based Optimizer to Find Good Plans

While the past parts of this training give you the knowledge of how to optimize SQL and control execution plans yourself, it is not realistic to manually tune every SQL statement yourself. This is exactly why the Cost Based Optimizer exists.

Unfortunately, Oracle's Cost Based Optimization process is not perfect; in some cases, because of the Optimizer's design limitations, sometimes bugs, but often because incorrectly configured optimizer environment and statistics collection setup.

This part of the training provides you the required understanding of CBO internals and SQL plan execution's cost estimation process. Armed with this knowledge you will learn how to put together a strategy for configuring Cost Based Optimizer environment and statistics collection correctly. We will look into different requirements by different workloads (such as OLTP vs. DSS systems).

Additionally, we will look into how to systematically work out the root causes of CBO row count and cost misestimates resulting in plan instability and bad performance.

Understanding CBO inputs

- SQL text
  - SQL of views accessed
  - Columns selected
  - Predicate conditions
  - Hints in SQL statements (or profiles/baselines)
- Table, index structure
  - Table columns, indexed columns and indexed column order
  - Table/index physical structure (cluster, index-organized table etc)
  - Partitioning structure and keys
- Table constraints
- Optimizer statistics, System statistics
- Sampled dynamic statistics, optimizer plan directives
- Session parameters (optimizer environment)
- Work area size used (pga_aggregate_target)

Understanding CBO outputs

- What is the Optimizer Cost?
- How does CBO try to come up with efficient execution plan?
- Common reasons for CBOs wrong decisions
- Avoiding underestimating row counts

Understanding optimizer's by-design limitations

- Cursor reuse, bind variables & bind peeking
- Handling "out of range" predicate conditions
- Handling data skew
- Handling imbalanced filter and join correlations
- Using dynamic sampling and adaptive features for join-correlation aware statistics
- Costing PL/SQL functions used in SQL statements

Good CBO configuration practices

- Setting up optimizer for majority of queries
- Stopping fine-tuning optimizer parameters

Good statistics gathering practices

- How to gather realistic system statistics
- Where to use histograms and where not to do so
- Where to use multi-column statistics
- Freezing statistics
- Adjusting statistics
- Copying statistics
- Relying on dynamic sampling
- Oracle 12.2 & 18c improvements

Advanced CBO troubleshooting

- Using the optimizer trace (event 10053) for optimizer problem troubleshooting
  - Summarizing join order reasoning from the optimizer trace file
  - Drilling into specific access path costing and root cause
- Troubleshooting cardinality misestimates
- Troubleshooting cost-based query transformation problems
- Dealing with optimizer bugs and enabling/disabling fixes