Oracle SQL Optimization - Intro

Aptean Ltd

Contents

1 Oracle SQL Optimization - Intro

2 Identifying Poorly Performing SQL in Oracle: A Developer's Guide

Author: Database Engineering Team

This technical guide explains practical, developer-focused techniques for identifying and diagnosing poorly performing SQL statements in Oracle databases. It covers built-in Oracle performance views and tools, shows example queries, and provides guidance on interpreting execution plans, wait events, and prioritization criteria to focus tuning efforts where they matter most.

2.1 Overview

Poorly performing SQL is one of the most common root causes of slow applications in Oracle environments. Identifying these statements early allows developers to reduce response times, lower database load, and improve overall system stability.

Oracle provides several built-in performance monitoring views and reports that make it possible to find high-impact SQL without third-party tools. This document focuses on practical usage of AWR, V$SQL, ASH, and SQL Monitor from a developer perspective.

2.2 Why SQL Performance Matters

Inefficient SQL can consume excessive CPU, memory, and I/O, directly impacting other sessions on the same database. A single bad query can degrade the performance of an entire application. From a development standpoint, understanding SQL performance helps:

Prevent scalability issues.

Reduce operational costs.

Shorten incident resolution time when performance problems occur in production.

2.3 Top Methods to Identify Poorly Performing SQL

Oracle offers multiple complementary tools to analyze SQL performance. Each tool answers different questions, such as which SQL uses the most resources over time or which SQL is slow right now.

Oracle Performance Tools and Use Cases

Tool	Primary Use Case	Typical Scenario
AWR (Automatic Workload Repository)	Historical analysis of top SQL by resource usage	Investigating performance issues that occurred in the past
V$SQL	Current and recent SQL execution statistics	Finding SQL with high CPU or elapsed time right now
ASH (Active Session History)	Session-level wait and activity analysis	Understanding where time is spent during slow periods
SQL Monitor	Detailed execution monitoring for individual SQL	Diagnosing long-running or parallel queries

2.3.1 Using AWR to Find High-Impact SQL

AWR reports rank SQL statements by metrics such as elapsed time, CPU time, logical reads, and physical reads. These rankings help identify SQL that contributes most to overall database load. Developers commonly start with the 'SQL ordered by Elapsed Time' or 'SQL ordered by CPU Time' sections of the AWR report to locate tuning candidates. +2

2.3.2 Using VSQLforReal?TimeAnalysis

TheVSQL view provides execution statistics for SQL statements currently in the shared pool. It is useful for identifying SQL with high average execution time or excessive buffer gets. Developers should focus on per-execution metrics rather than total values to avoid being misled by frequently executed but inexpensive SQL.

Example V$SQL Metrics

Metric	Description
ELAPSED_TIME / EXECUTIONS	Average elapsed time per execution in microseconds
BUFFER GETS / EXECUTIONS	Logical I/O per execution
DISK_READS / EXECUTIONS	Physical reads per execution

2.3.3 Using ASH to Understand Waits

Active Session History (ASH) samples active database sessions and records what they are waiting for. ASH is especially useful when performance problems are intermittent. By grouping ASH data by SQL_ID and wait event, developers can identify which SQL statements are blocked by I/O, locks, or CPU contention. +2

2.3.4 Using SQL Monitor for Long-Running SQL

SQL Monitor automatically captures execution details for SQL statements that run longer than a threshold or use parallel execution. It provides step-by-step execution progress, row counts, and time spent in each operation, making it ideal for diagnosing complex queries. +1

2.4 Sample Queries

The following examples illustrate how developers can query Oracle dynamic performance views to identify poorly performing SQL.

Purpose	Example Query
Find SQL with highest average elapsed time	`SELECT sql_id, executions, elapsed_time/executions avg_elapsed FROM vsql WHERE executions > 0 ORDERBY avgelapsed DESC`
Find SQL with highest buffer gets per execution	`SELECTsql id, buffergets/executionsavg gets FROM vsql WHERE executions > 0 ORDER BY avg_gets DESC`

2.5 Execution Plan Analysis

Execution plans explain how Oracle accesses data and joins tables. Poor performance often correlates with inefficient access paths or join methods. Developers should look for:

Full table scans on large tables.

Unexpected nested loop joins.

Significant differences between estimated and actual row counts.

2.6 Common Red Flags and Wait Events

Certain patterns frequently indicate SQL that requires tuning. These red flags can often be spotted directly in execution plans or performance views.

Typical SQL Performance Red Flags

Red Flag	Why It Matters
Full table scan on large tables	Causes excessive I/O and CPU usage
High buffer gets per execution	Indicates inefficient data access

Red Flag	Why It Matters
Large difference between estimated and actual rows	Leads to suboptimal execution plans
Frequent hard parsing	Increases CPU usage and latch contention

Wait events provide insight into why SQL is slow.

Common Oracle Wait Events

Wait Event	Typical Cause
db file sequential read	Single-block I/O, often index access
db file scattered read	Multi-block I/O, often full table scans
CPU + Wait for CPU	CPU contention due to inefficient SQL
enq: TX - row lock contention	Blocking due to concurrent DML

2.7 Prioritization Checklist

Not all slow SQL should be tuned first. Developers should prioritize based on business impact and resource consumption.

SQL Tuning Prioritization Checklist

Criterion	Question to Ask
Execution frequency	Is this SQL executed thousands of times per hour?
User impact	Does it affect critical user-facing functionality?
Resource usage	Does it consume significant CPU or I/O?
Ease of fix	Can indexing or query rewrite provide quick gains?