

Introduction to SQL Tuning

Chapter 2

GÜVEN GÜNEL (guven.gunel@ingram.com.tr) has a non-transferable license to use this Student Guide.

2

Introduction to SQL Tuning

ORACLE

Objectives

Objectives

After completing this lesson, you should be able to:

- Describe what attributes of a SQL statement can make it perform poorly
- List the Oracle tools that can be used to tune SQL
- List the tuning tasks

ORACLE

Objectives

This lesson gives you an understanding of the tuning process and the different components of an Oracle Database that may require tuning.

Reasons for Inefficient SQL Performance

Reasons for Inefficient SQL Performance

- Stale or missing optimizer statistics
- Missing access structures
- Suboptimal execution plan selection
- Poorly constructed SQL

ORACLE

Reasons for Inefficient SQL Performance

SQL statements can perform poorly for a variety of reasons:

- **Stale optimizer statistics:** SQL execution plans are generated by the cost-based optimizer (CBO). For CBO to effectively choose the most efficient plan, it needs accurate information on the data volume and distribution of tables and indexes referenced in the queries. Without accurate optimizer statistics, the CBO can easily be misled and generate suboptimal execution plans.
- **Missing access structures:** Absence of access structures, such as indexes, materialized views, and partitions is a common reason for poor SQL performance. The right set of access structures can improve SQL performance by several orders of magnitude.
- **Suboptimal execution plan selection:** The CBO can sometimes select a suboptimal execution plan for a SQL statement. This happens for most part because of incorrect estimates of some attributes of that SQL statement, such as its cost, cardinality, or predicate selectivity.
- **Poorly constructed SQL:** If the SQL statement is designed poorly, there is not much that the optimizer can do to improve its performance. A missing join condition leading to a Cartesian product, or the use of more expensive SQL constructs like `UNION` in place of `UNION ALL`, are just a couple of examples of inefficient SQL design.

These four main causes of poor SQL optimization can have a drastic impact on performance.

Note: Additional reasons for poor performance might be connected with hardware-related issues, such as memory, I/Os, CPUs, and so on.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a
non-transferable license to use this Student Guide.

Inefficient SQL: Examples

Inefficient SQL: Examples

- 1

```
SELECT COUNT(*) FROM products p
WHERE prod_list_price <
1.15 * (SELECT avg(unit_cost) FROM costs c
WHERE c.prod_id = p.prod_id)
```
- 2

```
SELECT * FROM job_history jh, employees e
WHERE substr(to_char(e.employee_id),2) =
substr(to_char(jh.employee_id),2)
```
- 3

```
SELECT * FROM orders WHERE order_id_char = 1205
```
- 4

```
SELECT * FROM employees
WHERE to_char(salary) = :sal
```
- 5

```
SELECT * FROM parts_old
UNION
SELECT * FROM parts_new
```

ORACLE

Inefficient SQL: Examples

The slide shows five examples of possibly poorly constructed SQL that could easily result in inefficient execution.

1. This is a common business question type. The query determines how many products have list prices less than 15% above the average cost of the product. This statement has a correlated subquery, which means that the subquery is run for every row found in the outer query. The query is better written as:

```
SELECT COUNT(*) FROM products p,
(SELECT prod_id, AVG(unit_cost) ac FROM costs
GROUP BY prod_id) c
WHERE p.prod_id = c.prod_id AND
p.prod_list_price < 1.15 * c.ac
```
2. This query applies functions to the join columns, restricting the conditions where indexes can be used. Use a simple equality, if you can. Otherwise, a function-based index may be necessary.
3. This query has a condition that forces implicit data type conversion; the ORDER_ID_CHAR column is a character type, and the constant is a numeric type. You should make the literal match the column type.

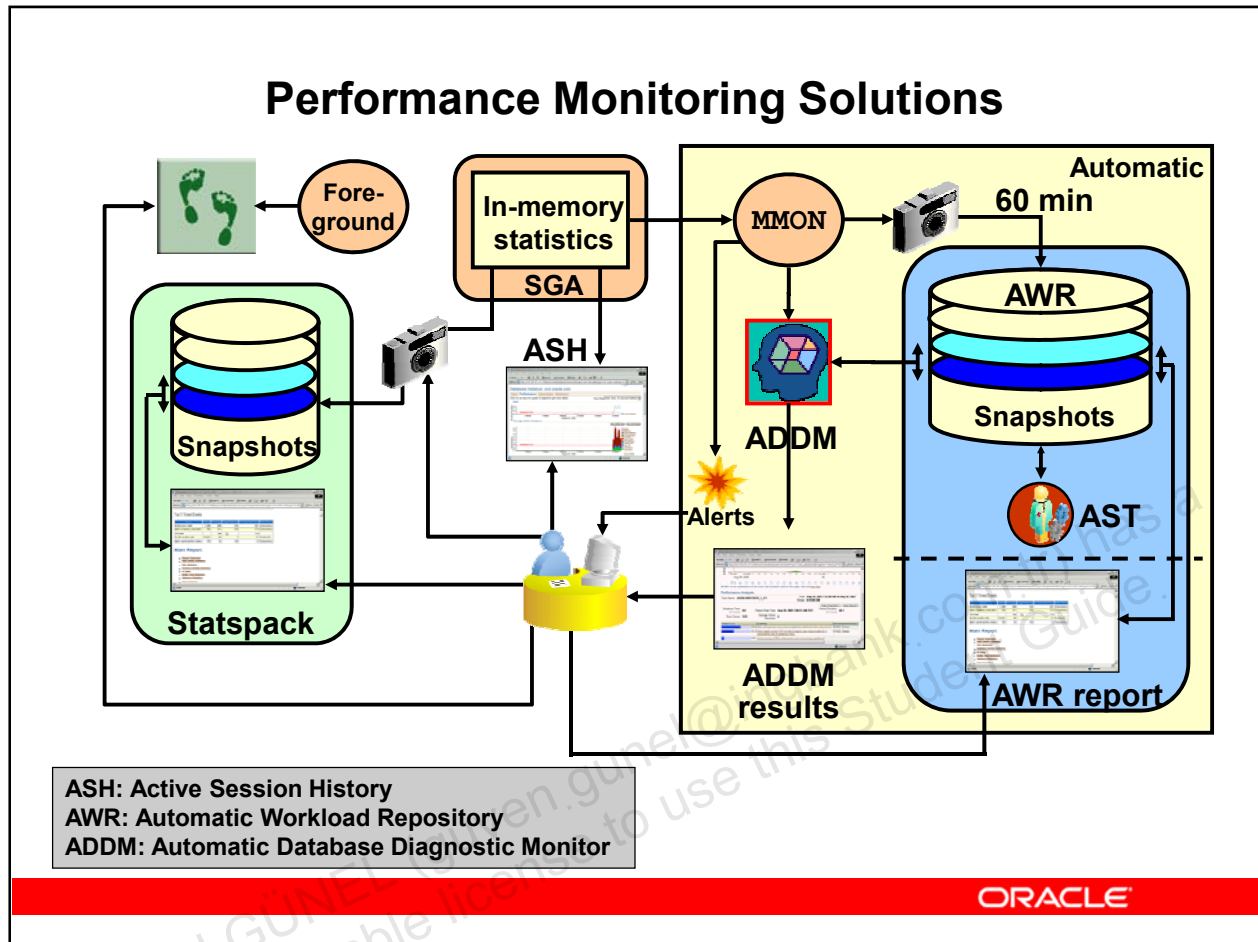
4. The fourth query uses a data type conversion function in it to make the data types match in the comparison. The problem here is that the `TO_CHAR` function is applied to the column value, rather than to the constant. This means that the function is called for every row in the table. It would be better to convert the literal once, and not convert the column. This is better queried as:

```
SELECT * FROM employees
WHERE salary = TO_NUMBER(:sal)
```

5. The `UNION` operator, as opposed to the `UNION ALL` operator, ensures that there are no duplicate rows in the result set. However, this requires an extra step, a unique sort, to eliminate any duplicates. If you know that there are no rows in common between the two `UNIONED` queries, use `UNION ALL` instead of `UNION`. This eliminates the unnecessary sort.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a non-transferable license to use this Student Guide.

Performance Monitoring Solutions



Performance Monitoring Solutions

Automatic Workload Repository (AWR): It collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. This data is both in memory and stored in the database. The gathered data can be displayed in both reports and views.

Active Session History (ASH): It provides sampled session activity in the instance. Active sessions are sampled every second and are stored in a circular buffer in SGA.

Snapshots: Snapshots are sets of historical data for specific time periods that are used for performance comparisons by ADDM. AWR compares the difference between snapshots to determine which SQL statements to capture based on the effect on the system load. This reduces the number of SQL statements that must be captured over time.

Automatic Database Diagnostic Monitor (ADDM)

In addition to the classical reactive tuning capabilities of previous releases, such as Statspack, SQL trace files, and performance views, Oracle Database 10g introduced new methodologies to monitor your database in two categories:

- **Proactive monitoring:**
 - Automatic Database Diagnostic Monitor (ADDM) automatically identifies bottlenecks within the Oracle Database. Additionally, working with other

manageability components, ADDM makes recommendations on the options available for fixing these bottlenecks.

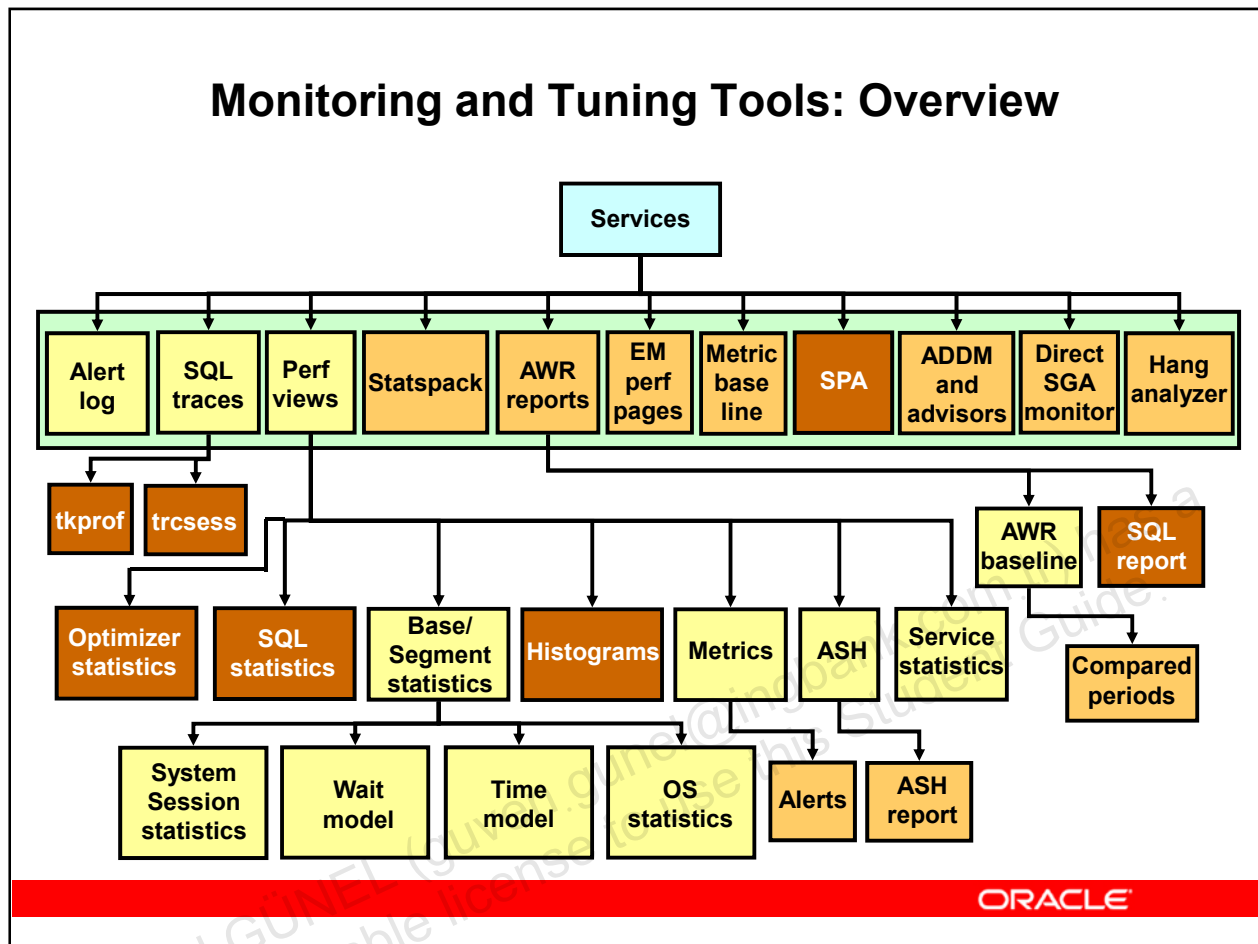
- Oracle Database 11g further automates the SQL tuning process by identifying problematic SQL statements, running SQL Tuning Advisor on them, and implementing the resulting SQL profile recommendations to tune the statement without requiring user intervention. This automation uses the AUTOTASK framework through a new task called Automatic SQL Tuning Task that runs every night by default.
- **Reactive monitoring:**
 - Server-generated alerts: The Oracle Database can automatically detect problematic situations. In reaction to a detected problem, the Oracle Database sends you an alert message with possible remedial action.
 - The Oracle Database has powerful new data sources and performance-reporting capabilities. Enterprise Manager provides an integrated performance management console that uses all relevant data sources. Using a drill-down method, you can manually identify bottlenecks with just a few mouse clicks.

New data sources are introduced to capture important information about your database's health—for example, new memory statistics (for current diagnostics) as well as statistics history stored in Automatic Workload Repository (AWR).

Note: Accessing Enterprise Manager or tools discussed here may require additional licenses and certain privileges generally reserved for database administrators.

GÜVEN GÜNEL (guven.gunel@bank.com.tr) has a non-transferable license to use this Student Guide.

Monitoring and Tuning Tools: Overview



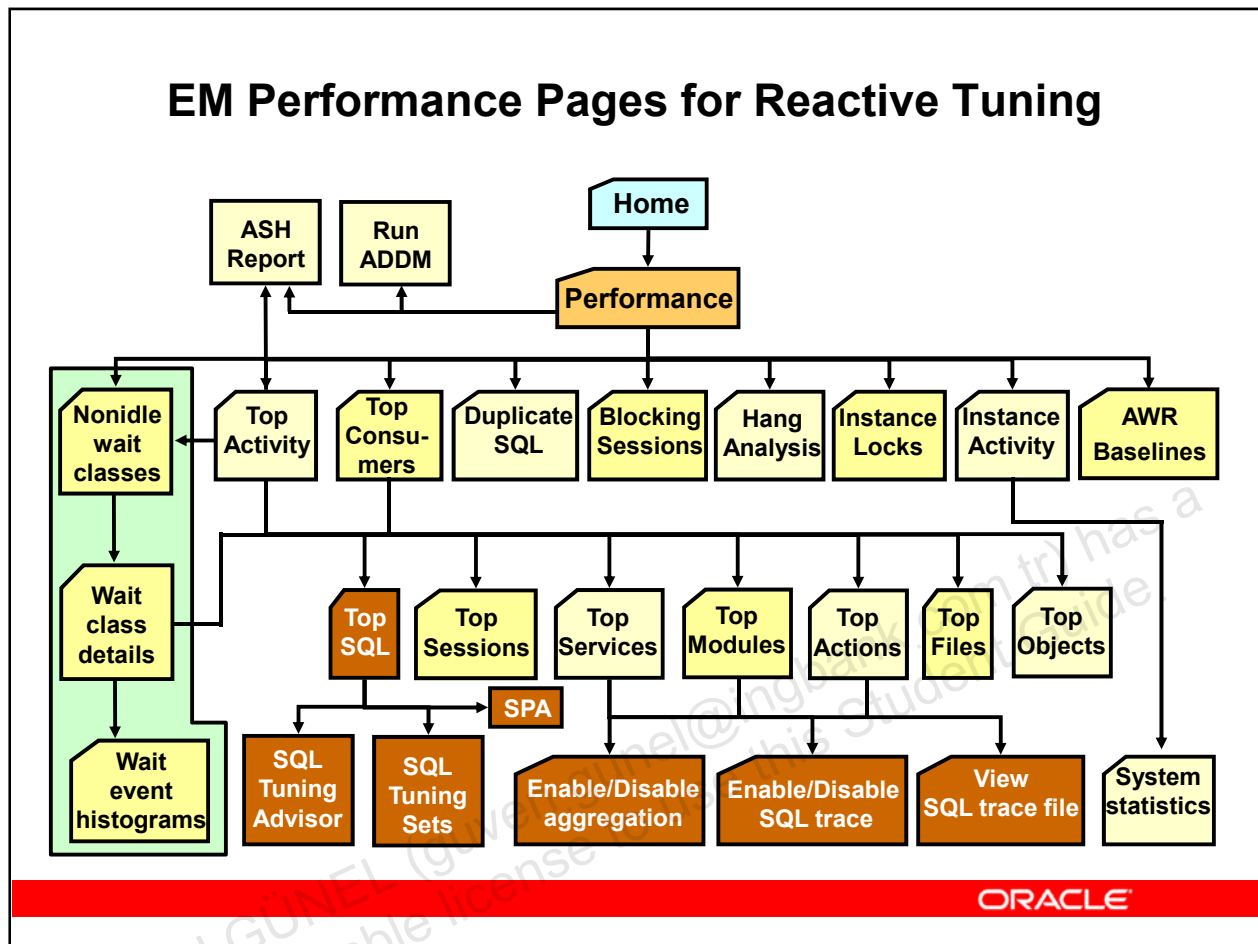
Monitoring and Tuning Tools: Overview

Since Oracle Database 10g, Release 2, you can generate SQL reports from AWR data (`$ORACLE_HOME/rdbms/admin/awrsqrpt.sql`), basically, the equivalent to `sqrepsql.sql` in Statspack.

Note

- SPA stands for SQL Performance Analyzer.

EM Performance Pages for Reactive Tuning



EM Performance Pages for Reactive Tuning

There are cases where real-time problem diagnosis must be performed. An irate user calls you, or you see a sudden spike in the activity of the system on the monitor. The Enterprise Manager (EM) Performance pages use the same data sources as AWR and ADDM to display information about the running of the database and the host system in a manner that is easily absorbed and allows for rapid manual drilldown to the source of the problem.

Tuning Tools: Overview

Tuning Tools: Overview

- Automatic Database Diagnostic Monitor (ADDM)
- SQL Tuning Advisor
- SQL Tuning Sets
- SQL Access Advisor
- SQL Performance Analyzer
- SQL Monitoring
- SQL Plan Management

ORACLE

Tuning Tools: Overview

Automatic Database Diagnostic Monitor: Continually analyzes the performance data that is collected from the database instance

SQL Tuning Advisor: Analyzes SQL statements that have been identified as problematic, in an effort to retune them. By default, this is an automated task. You can also, at any time, run the SQL Tuning Advisor on a specific SQL workload to look for ways to improve performance.

SQL Tuning Sets: Serve as a repository for sets of SQL statements. For example, the SQL Tuning Advisor can run against a workload that is represented by a SQL Tuning Set. They can even be transported from database to database, to perform analysis on different machines.

SQL Access Advisor: Analyzes a SQL statement, and provides advice on materialized views, indexes, materialized view logs, and partitions

SQL Performance Analyzer: Automates the process of assessing the overall effect of a change, such as upgrading a database or adding new indexes, on the full SQL workload by identifying performance divergence for each statement

SQL Monitoring: Enables you to monitor the performance of SQL statements while they execute

SQL Plan Management (SPM): Can be used to control execution plan evolution. By creating a SQL baseline, SPM will allow only approved execution plans to be used. Other plans

discovered by the optimizer will be stored in the SQL plan history, but will not be used until they are approved.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a
non-transferable license to use this Student Guide.

SQL Tuning Tasks: Overview

SQL Tuning Tasks: Overview

- Identifying high-load SQL
- Gathering statistics
- Generating system statistics
- Rebuilding existing indexes
- Maintaining execution plans
- Creating new index strategies

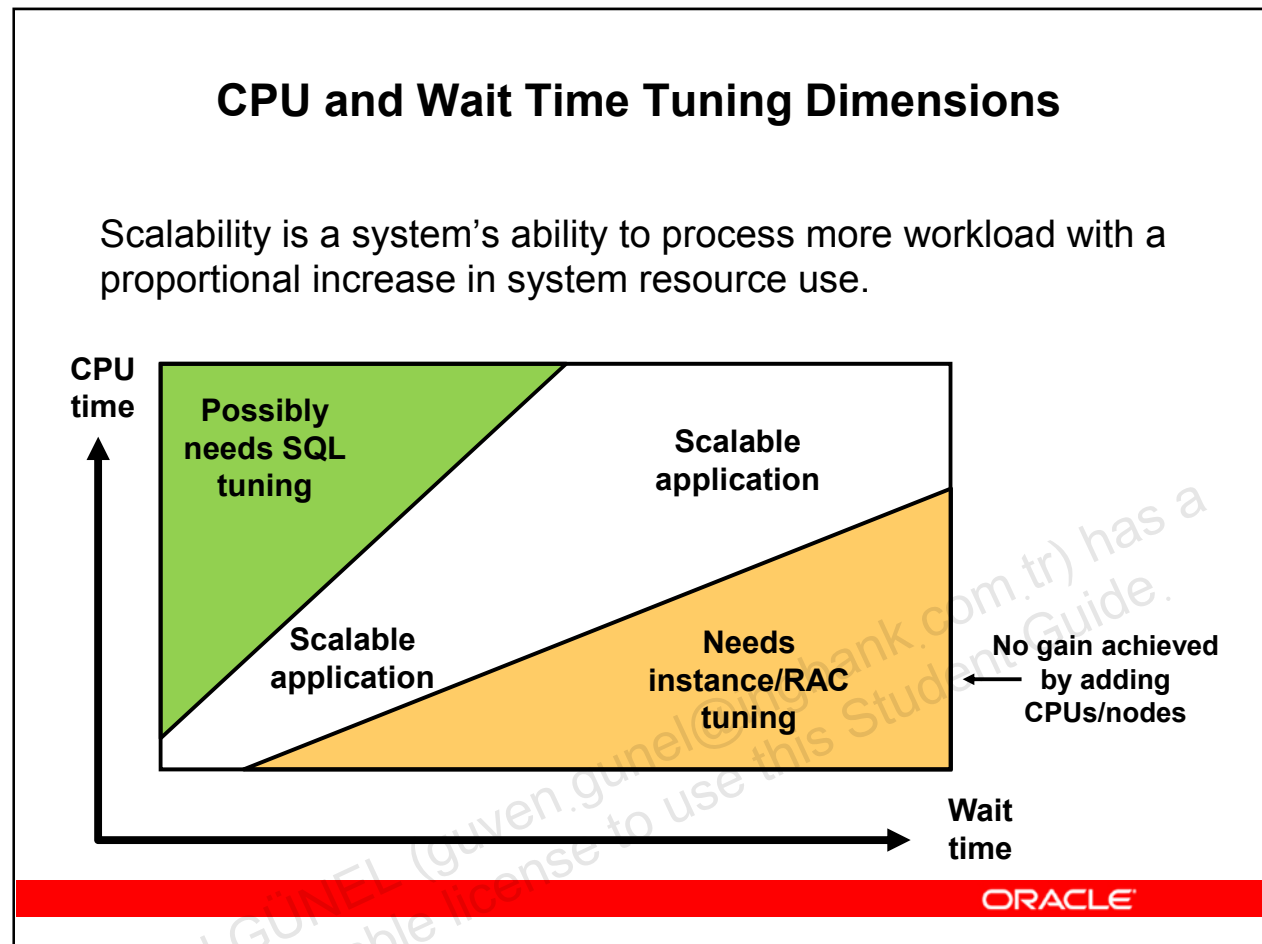
ORACLE

SQL Tuning Tasks: Overview

Many SQL tuning tasks should be performed on a regular basis. You may see a way to rewrite a `WHERE` clause, but it may depend on a new index being built. This list of tasks gives you a background of some important tasks that must be performed, and gives you an idea of what dependencies you may have as you tune SQL:

- Identifying high-load SQL statements is one of the most important tasks you should perform. The ADDM is the ideal tool for this particular task.
- By default, the Oracle Database gathers optimizer statistics automatically. For this, a job is scheduled to run in the maintenance windows.
- Operating system statistics provide information about the usage and performance of the main hardware components as well as the performance of the operating system itself.
- Often, there is a beneficial impact on performance by rebuilding indexes. For example, removing nonselective indexes to speed the data manipulation language (DML), or adding columns to the index to improve selectivity.
- You can maintain the existing execution plan of SQL statements over time by using stored statistics or SQL plan baselines.

CPU and Wait Time Tuning Dimensions



CPU and Wait Time Tuning Dimensions

When you tune your system, it is important that you compare the CPU time with the wait time of your system. By comparing CPU time with wait time, you can determine how much of the response time is spent on useful work and how much on waiting for resources potentially held by other processes.

As a general rule, the systems where CPU time is dominant usually need less tuning than the ones where wait time is dominant. On the other hand, high CPU usage can be caused by badly-written SQL statements.

Although the proportion of CPU time to wait time always tends to decrease as load on the system increases, steep increases in wait time are a sign of contention and must be addressed for good scalability.

Adding more CPUs to a node, or nodes to a cluster, would provide very limited benefit under contention. Conversely, a system where the proportion of CPU time does not decrease significantly as load increases can scale better, and would most likely benefit from adding CPUs or Real Application Clusters (RAC) instances if needed.

Note: AWR reports display CPU time together with wait time in the Top 5 Timed Events section, if the CPU time portion is among the top five events.

Scalability with Application Design, Implementation, and Configuration

Scalability with Application Design, Implementation, and Configuration

Applications have a significant impact on scalability.

- Poor schema design can cause expensive SQL that does not scale.
- Poor transaction design can cause locking and serialization problems.
- Poor connection management can cause unsatisfactory response times.

ORACLE

Scalability with Application Design, Implementation, and Configuration

Poor application design, implementation, and configuration have a significant impact on scalability. This results in:

- Poor SQL and index design, resulting in a higher number of logical input/output (I/O) for the same number of rows returned
- Reduced availability because database objects take longer to maintain

However, design is not the only problem. The physical implementation of the application can be the weak link, as in the following examples:

- Systems can move to production environments with poorly written SQL that cause high I/O.
- Infrequent transaction `COMMITs` or `ROLLBACKs` can cause long locks on resources.
- The production environment can use different execution plans than those generated in testing.
- Memory-intensive applications that allocate a large amount of memory without much thought for freeing the memory can cause excessive memory fragmentation.
- Inefficient memory usage places high stress on the operating virtual memory subsystem. This affects performance and availability.

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Common Mistakes on Customer Systems

Common Mistakes on Customer Systems

1. Bad connection management
2. Bad use of cursors and the shared pool
3. Excess of resources consuming SQL statements
4. Use of nonstandard initialization parameters
5. Poor database disk configuration
6. Redo log setup problems
7. Excessive serialization
8. Inappropriate full table scans
9. Large number of recursive SQL statements related to space management or parsing activity
10. Deployment and migration errors

ORACLE

Common Mistakes on Customer Systems

1. **Bad connection management:** The application connects and disconnects for each database interaction. This problem is common with stateless middleware in application servers. It has over two orders of magnitude impact on performance, and is not scalable.
2. **Bad use of cursors and the shared pool:** Not using cursors results in repeated parses. If bind variables are not used, there may be hard parsing of all similar SQL statements. This has an order of magnitude impact on performance, and it is not scalable. Use cursors with bind variables that open the cursor and execute it many times. Be suspicious of applications generating dynamic SQL.
3. **Bad SQL:** Bad SQL is SQL that uses more resources than appropriate for the application. This can be a decision support system (DSS) query that runs for more than 24 hours or a query from an online application that takes more than a minute. SQL that consumes significant system resources should be investigated for potential improvement. ADDM identifies high-load SQL and the SQL Tuning Advisor can be used to provide recommendations for improvement.
4. **Use of nonstandard initialization parameters:** These might have been implemented based on poor advice or incorrect assumptions. Most systems give acceptable performance using only the set of basic parameters. In particular, undocumented optimizer features can cause a great deal of problems that may require considerable investigation. Likewise, optimizer parameters set in the initialization parameter file can

override proven optimal execution plans. For these reasons, schemas, schema statistics, and optimizer settings should be managed together as a group to ensure consistency of performance.

5. **Getting database I/O wrong:** Many sites lay out their databases poorly over the available disks. Other sites specify the number of disks incorrectly because they configure disks by disk space and not by I/O bandwidth.
6. **Redo log setup problems:** Many sites run with too small redo log files. Small redo logs cause system checkpoints to continuously put a high load on the buffer cache and the I/O system. If there are very few redo logs, the archive cannot keep up, and the database waits for the archive process to catch up.
7. **Excessive serialization:** Serialization of data blocks in the buffer cache due to shortage of undo segments is particularly common in applications with large numbers of active users and a few undo segments. Use Automatic Segment Space Management (ASSM) and automatic undo management to solve this problem.
8. **Long full table scans:** Long full table scans for high-volume or interactive online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Long table scans, by nature, are I/O-intensive and not scalable.
9. **High amounts of recursive (sys) SQL:** Large amounts of recursive SQL executed by SYS could indicate that space management activities, such as extent allocations, take place. This is not scalable and impacts user response time. Use locally managed tablespaces to reduce recursive SQL due to extent allocation. Recursive SQL executed under another user ID is probably SQL and PL/SQL, so this is not a problem.
10. **Deployment and migration errors:** In many cases, an application uses too many resources because the schema owning the tables has not been successfully migrated from the development environment or from an older implementation. Examples of this are missing indexes or incorrect statistics. These errors can lead to suboptimal execution plans and poor interactive user performance. When migrating applications of known performance, export the schema statistics to maintain plan stability using the DBMS_STATS package.

Although these errors are not directly detected by ADDM, ADDM highlights the resulting high-load SQL.

Proactive Tuning Methodology

Proactive Tuning Methodology

- Simple design
- Data modeling
- Tables and indexes
- Using views
- Writing efficient SQL
- Cursor sharing
- Using bind variables

ORACLE

Proactive Tuning Methodology

Tuning usually implies fixing a performance problem. However, tuning should be part of the life cycle of an application, through the analysis, design, coding, production, and maintenance stages. The tuning phase is often left until the system is in production. At that time, tuning becomes a reactive exercise, where the most important bottleneck is identified and fixed.

The slide lists some of the issues that affect performance and that should be tuned proactively instead of reactively. These are discussed in more detail in the following slides.

Simplicity in Application Design

Simplicity in Application Design

- Simple tables
- Well-written SQL
- Indexing only as required
- Retrieving only required information

ORACLE

Simplicity in Application Design

Applications are no different from any other designed and engineered product. If the design looks right, it probably is right. This principle should always be kept in mind when building applications. Consider some of the following design issues:

- If the table design is so complicated that nobody can fully understand it, the table is probably designed badly.
- If SQL statements are so long and involved that it would be impossible for any optimizer to effectively optimize it in real time, there is probably a bad statement, underlying transaction, or table design.
- If there are many indexes on a table and the same columns are repeatedly indexed, there is probably a bad index design.
- If queries are submitted without suitable qualification (the `WHERE` clause) for rapid response for online users, there is probably a bad user interface or transaction design.

Data Modeling

Data Modeling

- Accurately represent business practices
- Focus on the most frequent and important business transactions
- Use modeling tools
- Appropriately normalize data (OLTP versus DW)

ORACLE

Data Modeling

Data modeling is important in successful relational application design. This should be done in a way that quickly and accurately represents the business practices. Apply your greatest modeling efforts to those entities affected by the most frequent business transactions. Use of modeling tools can then rapidly generate schema definitions and can be useful when a fast prototype is required.

Normalizing data prevents duplication. When data is normalized, you have a clear picture of the keys and relationships. It is then easier to perform the next step of creating tables, constraints, and indexes. A good data model ultimately means that your queries are written more efficiently.

Table Design

Table Design

- Compromise between flexibility and performance:
 - Principally normalize
 - Selectively denormalize
- Use Oracle performance and management features:
 - Default values
 - Constraints
 - Materialized views
 - Clusters
 - Partitioning
- Focus on business-critical tables

ORACLE

Table Design

Table design is largely a compromise between flexibility and performance of core transactions. To keep the database flexible and able to accommodate unforeseen workloads, the table design should be very similar to the data model, and it should be normalized to at least third normal form. However, certain core transactions can require selective denormalization for performance purposes.

Use the features supplied with Oracle Database to simplify table design for performance, such as storing tables prejoined in clusters, adding derived columns and aggregate values, and using materialized views or partitioned tables. Additionally, create check constraints and columns with default values to prevent bad data from getting into the tables.

Design should be focused on business-critical tables so that good performance can be achieved in areas that are the most used. For noncritical tables, shortcuts in design can be adopted to enable a more rapid application development. If, however, a noncore table becomes a performance problem during prototyping and testing, remedial design efforts should be applied immediately.

Index Design

Index Design

- Create indexes on the following:
 - Primary key (can be automatically created)
 - Unique key (can be automatically created)
 - Foreign keys (good candidates)
- Index data that is frequently queried (select list).
- Use SQL as a guide to index design.

ORACLE

Index Design

Index design is also a largely iterative process based on the SQL that is generated by application designers. However, it is possible to make a sensible start by building indexes that enforce foreign key constraints (to reduce response time on joins between primary key tables and foreign key tables) and creating indexes on frequently accessed data, such as a person's name. Primary keys and unique keys are automatically indexed except for the `DISABLE` `VALIDATE` and `DISABLE NOVALIDATE RELY` constraints. As the application evolves and testing is performed on realistic sizes of data, certain queries need performance improvements, for which building a better index is a good solution.

The following indexing design ideas should be considered when building a new index.

Appending Columns to an Index or Using Index-Organized Tables

One of the easiest ways to speed up a query is to reduce the number of logical I/Os by eliminating a table scan from the execution plan. This can be done by creating an index over all the columns of the table referenced by the query. These columns are the select list columns, `WHERE` clause columns, and any required join or sort columns. This technique is particularly useful in speeding up an online application's response times when time-consuming I/Os are reduced. This is best applied when testing the application with properly-sized data for the first time. The most aggressive form of this technique is to build an index-organized table (IOT).

Using Views

Using Views

- Simplifies application design
- Is transparent to the developer
- Can cause suboptimal execution plans

ORACLE

Using Views

Views can speed up and simplify application design. A simple view definition can mask data model complexity from the programmers whose priorities are to retrieve, display, collect, and store data. Views are often used to provide simple row and column-level access restrictions.

However, though views provide clean programming interfaces, they can cause suboptimal, resource-intensive queries when nested too deeply. The worst type of view use is creating joins on views that reference other views, which in turn reference other views. In many cases, developers can satisfy the query directly from the table without using a view. Because of their inherent properties, views usually make it difficult for the optimizer to generate the optimal execution plan.

SQL Execution Efficiency

SQL Execution Efficiency

- Good database connectivity
- Minimizing parsing
- Share cursors
- Using bind variables

ORACLE

SQL Execution Efficiency

An application that is designed for SQL execution efficiency must support the following characteristics:

Good database connection management: Connecting to the database is an expensive operation that is not scalable. Therefore, the number of concurrent connections to the database should be minimized as much as possible. A simple system, where a user connects at application initialization, is ideal. However, in a Web-based or multitiered application, where application servers are used to multiplex database connections to users, this can be difficult. With these types of applications, design efforts should ensure that database connections are pooled and not reestablished for each user request.

Good cursor usage and management: Maintaining user connections is equally important for minimizing the parsing activity on the system. Parsing is the process of interpreting a SQL statement and creating an execution plan for it. This process has many phases, including syntax checking, security checking, execution plan generation, and loading shared structures into the shared pool.

There are two types of parse operations:

- **Hard parsing:** A SQL statement is submitted for the first time, and no match is found in the shared pool. Hard parses are the most resource-intensive and are not scalable because they perform all the operations involved in a parse.

- **Soft parsing:** A SQL statement is submitted for the first time, and a match is found in the shared pool. The match can be the result of previous execution by another user. The SQL statement is shared, which is good for performance. However, soft parses are not ideal because they still require syntax and security checking, which consume system resources.

Because parsing should be minimized as much as possible, application developers should design their applications to parse SQL statements once and execute them many times. This is done through cursors. Experienced SQL programmers should be familiar with the concept of opening and reexecuting cursors.

Application developers must also ensure that SQL statements are shared within the shared pool. To do this, bind variables to represent the parts of the query that change from execution to execution. If this is not done, the SQL statement is likely to be parsed once and never reused by other users. To ensure that SQL is shared, use bind variables and do not use string literals with SQL statements.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a non-transferable license to use this Student Guide.

Writing SQL to Share Cursors

Writing SQL to Share Cursors

- Create generic code using the following:
 - Stored procedures and packages
 - Database triggers
 - Any other library routines and procedures
- Write to format standards (improves readability):
 - Case
 - White space
 - Comments
 - Object references
 - Bind variables

ORACLE

Writing SQL to Share Cursors

Applications can share cursors when the code is written in the same way characterwise (which allows the system to recognize that two statements are the same and thus can be shared), even if you use some special initialization parameters, such as `CURSOR_SHARING` discussed later in the lesson titled “Using Bind Variables.” You should develop coding conventions for SQL statements in ad hoc queries, SQL scripts, and Oracle Call Interface (OCI) calls.

Use generic shared code:

- Write and store procedures that can be shared across applications.
- Use database triggers.
- Write referenced triggers and procedures when using application development tools.
- Write library routines and procedures in other environments.

Write to format standards:

- Develop format standards for all statements, including those in PL/SQL code.
- Develop rules for the use of uppercase and lowercase characters.
- Develop rules for the use of white space (spaces, tabs, returns).

- Develop rules for the use of comments (preferably keeping them out of the SQL statements themselves).
- Use the same names to refer to identical database objects. If possible, prefix each object with a schema name.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a
non-transferable license to use this Student Guide.

Performance Checklist

Performance Checklist

- Set initialization parameters and storage options.
- Verify resource usage of SQL statements.
- Validate connections by middleware.
- Verify cursor sharing.
- Validate migration of all required objects.
- Verify validity and availability of optimizer statistics.

ORACLE

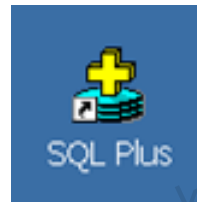
Performance Checklist

- Set the minimal number of initialization parameters. Ideally, most initialization parameters should be left at default. If there is more tuning to perform, this shows up when the system is under load. Set storage options for tables and indexes in appropriate tablespaces.
- Verify that all SQL statements are optimal and understand their resource usage.
- Validate that middleware and programs that connect to the database are efficient in their connection management and do not log on and log off repeatedly.
- Validate that the SQL statements use cursors efficiently. Each SQL statement should be parsed once and then executed multiple times. This happens mostly when bind variables are not used properly and the `WHERE` clause predicates are sent as string literals.
- Validate that all schema objects are correctly migrated from the development environment to the production database. This includes tables, indexes, sequences, triggers, packages, procedures, functions, Java objects, synonyms, grants, and views. Ensure that any modifications made in testing are made to the production system.
- As soon as the system is rolled out, establish a baseline set of statistics from the database and operating system. This first set of statistics validates or corrects any assumptions made in the design and rollout process.

Development Environments: Overview

Development Environments: Overview

- SQL Developer
- SQL*Plus



ORACLE

PL/SQL Development Environments

SQL Developer

This course has been developed using Oracle SQL Developer as the tool for running the SQL statements discussed in the examples in the slide and the practices.

- SQL Developer is shipped with Oracle Database 11g Release 2, and is the default tool for this class.

SQL*Plus

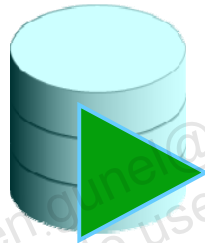
The SQL*Plus environment may also be used to run all SQL commands covered in this course.

Note: See Appendix C titled “Using SQL Developer” for information about using SQL Developer, including simple instructions on installing version 2.1.

What Is Oracle SQL Developer?

What Is Oracle SQL Developer?

- Oracle SQL Developer is a free graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema using standard Oracle database authentication.
- You will use SQL Developer in this course.
- Appendix C contains details on using SQL Developer



SQL Developer

ORACLE

What Is Oracle SQL Developer?

Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and maintain stored procedures, test SQL statements, and view optimizer plans.

SQL Developer, the visual tool for database development, simplifies the following tasks:

- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports


You can connect to any target Oracle database schema using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

Appendix C

Appendix C of this course provides an introduction on using the SQL Developer interface. It also provides information about creating a database connection, interacting with data using SQL and PL/SQL, and so on.

Coding PL/SQL in SQL*Plus

Coding PL/SQL in SQL*Plus



```
Terminal
SQL*Plus: Release 11.2.0.0.2 Beta on Thu May 28 21:20:35 2009

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Enter user-name: ora41
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> set serveroutput on
SQL> create or replace procedure hello is
2 begin
3 dbms_output.put_line('Hello Class!');
4 end;
5 /

Procedure created.

SQL> execute hello
Hello Class!

PL/SQL procedure successfully completed.

SQL>
```

ORACLE

Coding PL/SQL in SQL*Plus

Oracle SQL*Plus is a command-line interface that enables you to submit SQL statements and PL/SQL blocks for execution and receive the results in an application or a command window.

SQL*Plus is:

- Shipped with the database
- Accessed from an icon or the command line

When coding PL/SQL subprograms using SQL*Plus, remember the following:

- You create subprograms by using the `CREATE SQL` statement.
- You execute subprograms by using either an anonymous PL/SQL block or the `EXECUTE` command.
- If you use the `DBMS_OUTPUT` package procedures to print text to the screen, you must first execute the `SET SERVEROUTPUT ON` command in your session.

Note

- To launch SQL*Plus in the Linux environment, open a Terminal window and enter the `sqlplus` command.

- For more information about how to use SQL*Plus, see the SQL*Plus User's Guide and Reference.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a non-transferable license to use this Student Guide.

Quiz

Quiz

_____ automatically identifies bottlenecks within Oracle Database and makes recommendations on the options available for fixing these bottlenecks.

- a. ASH
- b. AWR
- c. ADDM
- d. Snapshots

ORACLE

Answer: c

Quiz

Quiz

Normalizing data results in a good data model, which ultimately means that your queries are written more efficiently.

- a. True
- b. False

ORACLE

Answer: a

Quiz

Quiz

Views should be used carefully because, though they provide clean programming interfaces, they can cause suboptimal, resource-intensive queries when nested too deeply.

- a. True
- b. False

ORACLE

Answer: a

Quiz

Quiz

Identify the characteristics that must be supported by an application designed for SQL execution efficiency.

- a. Use concurrent connections to the database.
- b. Use cursors so that SQL statements are parsed once and executed multiple times.
- c. Use bind variables.

ORACLE

Answer: b, c

Summary

Summary

In this lesson, you should have learned how to:

- Describe what attributes of a SQL statement can make it perform poorly
- List the Oracle tools that can be used to tune SQL
- List the tuning tasks

ORACLE

Practice 2: Overview

Practice 2: Overview

This practice covers the following topics:

- Rewriting queries for better performance
- Rewriting applications for better performance

ORACLE

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a
non-transferable license to use this Student Guide.