Oracle 11g
Advanced
SQL Programming

Student Workbook
Oracle 11g Advanced SQL Programming

Brian Peasland, Robert Roselius, and Robert Seitz

Published by ITCourseware, LLC., 7245 South Havana Street, Suite 100, Centennial, CO 80112

Editor: Jan Waleri

Editorial Assistant: Dana Howell

Special thanks to: Many instructors whose ideas and careful review have contributed to the quality of this workbook and the many students who have offered comments, suggestions, criticisms, and insights.

Copyright © 2013 by ITCourseware, LLC. All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photo-copying, recording, or by an information storage retrieval system, without permission in writing from the publisher. Inquiries should be addressed to ITCourseware, LLC., 7245 South Havana Street, Suite 100, Centennial, Colorado, 80112. (303) 302-5280.

All brand names, product names, trademarks, and registered trademarks are the property of their respective owners.
# Contents

Chapter 1 - Course Introduction ............................................................................................................. 9
   Course Objectives .......................................................................................................................... 10
   Course Overview ........................................................................................................................... 12
   Using the Workbook ...................................................................................................................... 13
   Suggested References ..................................................................................................................... 14

Chapter 2 - Database Design Concepts ................................................................................................ 17
   Relational Databases ....................................................................................................................... 18
   The Relational Model ...................................................................................................................... 20
   Relational Operations ...................................................................................................................... 22
   The Database Design Process ......................................................................................................... 24
   Normalization ................................................................................................................................. 26
   Second and Third Normal Forms .................................................................................................... 28
   Other Normal Forms ...................................................................................................................... 30
   Applications for Relational Databases .............................................................................................. 32
   Labs ............................................................................................................................................... 34

Chapter 3 - SQL Subqueries ................................................................................................................ 37
   Overview Of Subqueries ................................................................................................................. 38
   Correlated Subqueries .................................................................................................................... 40
   EXISTS Clause vs. IN Clause ............................................................................................................ 42
   Quantified Comparisons: ANY and ALL ......................................................................................... 44
   Scalar Subquery Expression ............................................................................................................ 46
   Subqueries and DML Statements ....................................................................................................... 48
   Inline Views .................................................................................................................................... 50
   Top-N and Bottom-N analysis .......................................................................................................... 52
   CREATE TABLE and Subqueries ....................................................................................................... 56
   Labs ............................................................................................................................................... 58
## Oracle11g Advanced SQL Programming

### Chapter 4 - Hierarchical Queries

- Hierarchical Data .......................................................... 62
- Self Joins ................................................................. 64
- Hierarchical Query ....................................................... 66
- Hierarchical Pseudocolumns ......................................... 68
- SYS_CONNECT_BY_PATH ............................................... 70
- Recursive Query Factoring ........................................... 72
- Labs ............................................................................ 74

### Chapter 5 - Object Types

- Object-Oriented Programming .................................... 78
- Oracle's Object Relational Model ................................. 80
- Creating Object Types .................................................. 82
- Querying Object Types .................................................. 84
- DML with Object Types ................................................ 86
- Object Methods ........................................................... 88
- Object Views ............................................................... 90
- VARRAYs .................................................................. 92
- Nested Tables ............................................................. 94
- Labs ............................................................................ 96

### Chapter 6 - Times, Dates, and Strings

- Datetime Fields ........................................................... 100
- Dates and Timestamps .................................................. 102
- Intervals ................................................................... 104
- Date and Interval Literals .............................................. 106
- Date Arithmetic .......................................................... 108
- Date Functions ........................................................... 110
- Character Types .......................................................... 112
- Session and Database Parameters .............................. 114
- REGEXP Functions ...................................................... 116
- Regular Expressions Supported by REGEXP ................. 118
- Applying REGEXP Functions ...................................... 120
- Labs ............................................................................ 122
# Oracle11g Advanced SQL Programming

**Chapter 7 - Temporary Tables** ................................................................. 125
- Undo and Redo ....................................................................................... 126
- Temporary Tables Defined .................................................................... 128
- Data Lifetime — Transaction vs. Session ............................................ 130
- Creating Temporary Tables ................................................................. 132
- Managing Temporary Tables ............................................................... 134
- Storage of Temporary Tables .............................................................. 136
- Effects of DML and TRUNCATE ........................................................... 138
- Labs ....................................................................................................... 140

**Chapter 8 - SQL Tuning Tools** ............................................................ 143
- Automated Statistics Gathering .......................................................... 144
- The DBMS_STATS Package ................................................................. 146
- SQL Tuning Advisor ............................................................................ 148
- SQL Tuning Sets .................................................................................. 150
- SQL Access Advisor ............................................................................ 152
- Retrieving Execution Plans ................................................................. 154
- EXPLAIN PLAN ..................................................................................... 156
- Using DBMS_XPLAN ............................................................................ 158
- Interpreting Explain Plan Results ......................................................... 160
- SQL Trace .............................................................................................. 162
- TKPROF .................................................................................................. 164
- Labs ....................................................................................................... 166

**Chapter 9 - SQL Tuning** ..................................................................... 171
- Tuning Goals ........................................................................................ 172
- The Optimizer ...................................................................................... 174
- Optimizer Statistics ............................................................................. 176
- Identifying SQL to Tune ....................................................................... 178
- Optimizer Hints ................................................................................... 180
- Optimizer Goal Hints .......................................................................... 182
- Access Path Hints ................................................................................ 184
- Join Hints ............................................................................................. 186
- Additional Hints ................................................................................... 188
- SQL Plan Management ........................................................................ 190
- Labs ....................................................................................................... 192
Chapter 10 - Indexes .......................................................................................................................... 195

Indexes ........................................................................................................................................ 196
B-tree and Composite Indexes ...................................................................................................... 198
Reverse Key and Unique Indexes ................................................................................................. 200
Function-Based Indexes ............................................................................................................... 202
Bitmap Indexes ............................................................................................................................. 204
Index-Organized Tables ................................................................................................................ 206
Managing Indexes ......................................................................................................................... 208
Labs ............................................................................................................................................. 210

Chapter 11 - Oracle Analytic Functions ............................................................................................... 213

ROLLUP ..................................................................................................................................... 214
CUBE .......................................................................................................................................... 216
Grouping Sets ............................................................................................................................... 218
Analytic Functions ........................................................................................................................ 220
OVER, PARTITION BY, and ORDER BY .................................................................................. 222
Windowing ................................................................................................................................... 224
RANK ......................................................................................................................................... 226
Modeling ...................................................................................................................................... 228
Model Clauses ............................................................................................................................. 230
Labs ............................................................................................................................................. 232

Chapter 12 - Data Warehouse Features .............................................................................................. 235

Partitioned Tables ......................................................................................................................... 236
Partitioning Methods ..................................................................................................................... 238
Partition Pruning and Partition-wise Joins ....................................................................................... 240
Bitmap Indexes ............................................................................................................................. 242
Materialized Views ....................................................................................................................... 244
Creating Materialized Views ......................................................................................................... 246
Refreshing Materialized Views ...................................................................................................... 248
The MERGE Statement ................................................................................................................ 250
Multi-table INSERT Statements .................................................................................................... 252
Parallel Statements ....................................................................................................................... 254
Labs ............................................................................................................................................. 256
Chapter 13 - Formatting Reports with SQL*Plus ................................................................. 259

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Formatting</td>
<td>260</td>
</tr>
<tr>
<td>Computations</td>
<td>262</td>
</tr>
<tr>
<td>SQL*Plus Options for Formatting</td>
<td>264</td>
</tr>
<tr>
<td>Saving the Output</td>
<td>266</td>
</tr>
<tr>
<td>Data Extraction with SQL*Plus</td>
<td>268</td>
</tr>
</tbody>
</table>

Solutions ......................................................................................................................... 271

Index ...................................................................................................................................... 301
Chapter 1 - Course Introduction
Course Objectives

- Apply the basic theory behind relational database design.
- Contribute to all phases of database design and development.
- Use all aspects of subqueries.
- Apply Oracle's features for querying hierarchical data models.
- Use Oracle's Object-Relational Model.
- Create object types.
- Use Oracle's collection types in SQL.
- Select appropriate date-related datatypes for your applications.
- Use Oracle's regular expression SQL functions to perform pattern matching and string manipulation.
- Create and manage temporary tables.
- Establish goals in SQL tuning to improve performance.
- Use Oracle Database 11g's tuning tools.
- Describe how indexes are used in RDBMSs, and use them effectively.
- Use the various analytic functions provided by Oracle to perform sophisticated analysis.
- Use SQL*Plus to format reports and extract data.
Course Overview

🌟 **Audience**: Application developers, database administrators, system administrators, and users who write applications and procedures that access an Oracle11g database.

🌟 **Prerequisites**: *Oracle11g SQL Programming*

🌟 **Classroom Environment**:

- A workstation per student.
- Oracle Database 11g, Release 2 with Oracle's Sample Schemas installed.
Using the Workbook

This workbook design is based on a page-pair, consisting of a Topic page and a Support page. When you lay the workbook open flat, the Topic page is on the left and the Support page is on the right. The Topic page contains the points to be discussed in class. The Support page has code examples, diagrams, screen shots and additional information. Hands On sections provide opportunities for practical application of key concepts. Try It and Investigate sections help direct individual discovery.

In addition, there is an index for quick look-up. Printed lab solutions are in the back of the book as well as on-line if you need a little help.
Suggested References


http://www.oracle.com/technetwork/dbadev/
http://asktom.oracle.com/
http://www.dbasupport.com/
http://www.orafaq.com/wiki/
http://www.quest-pipelines.com/
http://tahiti.oracle.com/
Your single most important reference is the SQL Reference book, which is part of the Oracle Database Online Documentation. You may have received this on CD-ROM with your Oracle distribution. If not, you can access it online at Oracle's website. This is the official, complete description of Oracle's implementation of SQL. It includes many examples and discussions.

An easy way to find it is to go to:

http://tahiti.oracle.com/

Find the documentation for your version of Oracle. Locate the SQL Reference and open the HTML table of contents.

If you have web access in the classroom, open a browser now and find the SQL Language Reference. Set a browser bookmark and have the SQL Reference at hand throughout this class.

Other essential Oracle Documentation Library bookmarks for Oracle developers:

Reference - System/Session parameters, Data Dictionary
Error Messages
Advanced Application Developer's Guide
Performance Tuning Guide - Indexes, tuning tools and techniques
Database Concepts - Oracle architecture, details of locking, transactions, memory and processes, etc.
PL/SQL Packages and Types Reference - Oracle supplied packages

Additional books related to material covered in this class:
Object-Relational Developer's Guide - Object and Collection types
SecureFiles and Large Objects Developer's Guide - BLOBs, CLOBs, BFILEs
Data Warehousing Guide - analytic and modeling functions, materialized views, ETL, etc.
Chapter 2 - Database Design Concepts

Objectives

* Apply the basic theory behind relational database design.
* Determine the data model’s entities and their attributes.
* Normalize tables in a relational database design.
* Categorize the operations a database system performs on data.
* Contribute to all phases of database design and development.
Relational Databases

- A Relational Database accepts and presents data according to rules based on the Relational Model.
- A Relational Database Management System (RDBMS) is the software that implements this capability.
- The Relational Model describes a particular way of representing data and constructing expressions to operate upon it.
  - Alternative models include the Hierarchical and Network models.
  - The Relational Model, first published by Dr. E. F. Codd in 1970, bases itself on set theory and has several goals:
    - Description of data independent of machine representation.
    - Independence of applications and users from data representation.
    - Provision of a high-level data language, based on predicate calculus.
    - Ability to interact with data independent of ordering, indexing, and access paths.
- In 1985, Dr. Codd published what are now known as Codd's Twelve Rules, describing the minimum requirements for a system to qualify as an RDBMS.
  - The Relational Model remains the predominant model for large databases.
  - Very few RDBMS products faithfully implement all aspects of the Relational Model.
  - Oracle, like most RDBMS products, provides for a practical subset of Codd's Rules and formal relational algebra.
Codd's Twelve (really, thirteen) Rules:

1. All information is represented as column values in rows of tables.

2. Each item of information is accessible by identifying a table, a column in that table, and the primary key of the row that contains the item.

3. The intersection of a specific row and column might have missing or inapplicable information (a "null value").

4. The entire definition of the database is available in a set of tables, which are a database in their own right (a System Catalog or Data Dictionary).

5. The system provides a single, comprehensive language for defining the structure of the database (Data Definition Language (DDL)) and manipulating and querying the data in it (Data Manipulation Language (DML)).

6. A view (virtual table) that is theoretically updatable must be actually updatable.

7. Users can identify a set of rows for insertion, updating, or deletion, as well as for retrieval.

8. Users and applications are not affected by changes in the underlying physical storage format of the language.

9. Users and applications are not affected by changes in the underlying logical storage of the data that do not actually remove information.

10. Integrity constraints are enforced by the system, not by users or applications.

11. Users and applications are not affected by changes in the physical distribution of storage of the data.

12. The system provides no means of subverting the rules and constraints of the system.

And Rule 0:
0. The system maintains itself through its relational abilities.
The Relational Model

The Relational Model applies specific mathematical techniques (relational algebra) and logic (predicate calculus) to the problem of shared access to large amounts of computer data.

The Relational Model is the source of much of the terminology underlying SQL and SQL databases.

- The fundamental element is the domain — a datatype (NUMBER: the domain consisting of all possible number values representable as an Oracle NUMBER).
  - An attribute consists of an attribute name and its type (its domain).
  - An attribute value consists of an attribute and a value of that attribute's type.

- A tuple is a set of attribute values.

- A relation consists of:
  - Its heading — a set of attributes (their names and types).
  - Its body — a specific set of tuples, each consisting of attribute values.
Relations, written in mathematical form:

- Company(Warehouse)
- Warehouse(WarehouseID, Name, Description, LocationID)
- Customer(CustomerID, Name, Street, City, State, Country, PostalCode, Email)
- PhoneNumber(CustomerID, PhoneType, Number)

To get an idea of how the mathematical aspects of the Relational Model ... er, relate ... to SQL databases, we can observe a loose correspondence between components of each:

<table>
<thead>
<tr>
<th>Relational Model</th>
<th>SQL Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Datatype</td>
</tr>
<tr>
<td>Attribute</td>
<td>A column definition</td>
</tr>
<tr>
<td>Relation</td>
<td>A table or result set</td>
</tr>
<tr>
<td>Tuple</td>
<td>A row in a table or result set</td>
</tr>
<tr>
<td>Unary relation</td>
<td>Table or result set with one column (e.g., the DUAL table)</td>
</tr>
<tr>
<td>Binary relation</td>
<td>Table or result set with two columns</td>
</tr>
<tr>
<td>Ternary relation</td>
<td>Table or result set with three columns</td>
</tr>
<tr>
<td>Arity</td>
<td>Number of columns in table or result set</td>
</tr>
<tr>
<td>Cardinality</td>
<td>Number of rows in a table or result set</td>
</tr>
<tr>
<td>Selection</td>
<td>... WHERE ...</td>
</tr>
<tr>
<td>Projection</td>
<td>SELECT DISTINCT ...</td>
</tr>
<tr>
<td>Cartesian product</td>
<td>SELECT ... FROM a, b;</td>
</tr>
<tr>
<td>Set union</td>
<td>SELECT ... UNION SELECT ...</td>
</tr>
<tr>
<td>Set difference</td>
<td>SELECT ... MINUS SELECT ...</td>
</tr>
<tr>
<td>Candidate key</td>
<td>Any of the primary or unique keys of a table</td>
</tr>
</tbody>
</table>

Note:
This is NOT intended to be a rigorous comparison of the Relational Model with SQL databases!
Relational Operations

Relational Algebra defines a few basic operations for a relation, as well as several operations built on the basic ones.

- **Selection** — The set of tuples in a relation, for which a logical proposition is true.
- **Projection** — The set of tuples consisting of the values for some of the attributes in a relation.
- **Cartesian Product** — The set of all combinations of tuples from two relations.
- **Set Union** — All tuples belonging to either, or both, of two relations.
- **Set Difference** — All tuples belonging to one relation, which are not also in the second of two relations.
The degree to which an RDBMS supports true relational algebra can be debated. All would agree that Oracle, like nearly every other RDBMS, does not fully support the Relational Model. The same is true for SQL itself. This does not necessarily detract from the practicality and power of the RDBMS or of SQL. However, we often refer to RDBMSs as "SQL databases" rather than "relational databases" as a reminder of their differences.

Some differences between the formal Relational Model (RM) and real-world RDBMSs include:

- The RM does not allow NULL values.
- Tuples are unordered; in SQL, column order is often significant (SELECT *, INSERT, etc.)
- Duplicate tuples are not allowed in a relation; duplicates can occur in SQL (in non-DISTINCT result sets, in tables with no unique or primary key, in a UNION ALL, etc.)
- In the RM, every attribute must be named; SQL allows unnamed attributes (SELECT 2+2, ...)

There are others, but you get the idea. It is not necessary to study relational algebra and set theory in order to advance your SQL and database design skills. However, some familiarity with the theoretical underpinnings of the system and language we work with can provide helpful insights.

It is natural to suppose that the term "Relational Database" refers to the relationships (foreign keys, joins) between the tables in the database. Actually, the term refers to the relational model, in which the "relation" is the set of attributes — a table.
The Database Design Process

One of the earliest phases of database design is development of the Conceptual Model.

- End users, customers, functional and process experts, managers, and subject-matter experts (both in the subject business, and in any existing legacy systems) are among those who can help the database designer during this process.

- During this phase, the database designer helps identify entities, their attributes, and dependencies and associations between them.
  - The cardinality (one-to-one, one-to-many, many-to-many) of associations also become apparent.

- Entity-Relationship diagrams help document this phase.

Development of the Logical Model from the Conceptual Model provides concrete mapping of conceptual elements into specific database structures (tables, columns, constraints, etc.)

- Normalization of tables typically occurs during this phase.

Finally, a Physical Model specifies how the database will be stored on a computer.

- Tablespaces, and their datafiles and storage parameters.

- Physical requirements of the database host system.

- Indexes, instance parameters, access methods, etc.
There are many variations on the process of database design, of varying degrees of formality, detail, and discipline. It's not uncommon for an experienced developer to design and develop the conceptual, logical, and physical model on the fly for a modest application database (say, a simple web application or the like). An organization investing in a large-scale system will (or should) employ a well-defined process, facilitated by database design experts, perhaps in concert with other analysis and design processes (such as Object-Oriented Analysis and Design (OOAD)).

One simple exercise for beginning the Conceptual Design is to start with a textual description of the application or database domain. In the text, start by picking out the nouns — these are potential entities and attributes. The verbs can indicate relationships. For example:

The company maintains warehouses in several locations to fulfill customer needs. Each warehouse has a warehouse identification number, name, facility description, and location identification number.

Each customer has an identification number. Customer records include customer name, street name, city or state, country, phone numbers (up to five phone numbers for each customer), and postal code. Some customers place orders through the Internet, so e-mail addresses are also recorded.

"Has-a" and "is-a-part-of" constructs can indicate entity attributes. Action verbs ("...customers place orders...") can indicate relationships between entities.
Normalization

* Normalization defines a set of incremental tests to assure a database design will be free of redundancy and associated data anomalies.

- A database (or subset of its tables) for which a certain level of normalization is true is said to be in that *normal form*.
- Most database designers try to achieve at least third normal form (3NF) with their designs, and perhaps higher (or lower) levels for certain tables.

* To be in first normal form (1NF), several things must be true of a table:

- Each row is unique.
- For each row, each column contains at most one atomic value.
  - An atomic value cannot be decomposed into smaller values — for example, a *fullname* attribute is non-atomic, since it can be decomposed into *firstname* and *lastname* attributes.
- For each row, each column value is non-repeating.
  - For example, a single *Customer* record cannot contain the list of that customer's *Orders*.
  - Redefine a repeating attribute as a separate entity.

* A table in 1NF has a defined primary key.

- Like all other attributes, a primary key column is non-decomposable.
- A table may have a composite primary key — in which the combination of values of the primary key columns uniquely identifies each row.
Normalization is a process of decomposing a table into one or more tables. There are multiple levels of normalization that can be applied. Each higher level of normalization adds more constraints onto the previous. Each normal form is defined by a set of rules that prohibit data redundancy. This prevents data inconsistencies on data updates, resulting in a more stable database.

The higher you go in normal forms, the more tables you will have to navigate to find your information resulting in slower performance. Though there are multiple levels of normalization a database can undergo, be careful not to compromise your performance needs by over-normalization.

The name, courses, grades, and semesters attributes are not atomic. The student table is put into 1NF by splitting name and creating an additional table: transcript.
SECOND AND THIRD NORMAL FORMS

A table is in second normal form (2NF) if:

- The table is in 1NF.
- Each attribute depends on the entire primary key.
  - A 1NF table that does not have a composite primary key is already in 2NF.

A table is in third normal form (3NF) if:

- The table is in 2NF.
- Each non-key attribute depends only on the primary key.
  - A 3NF table is a 2NF table with no non-key attribute dependent on another non-key column.
The **transcript** table has a composite primary key consisting of the following attributes: `ssn`, `course_id`, `semester`, `year`.

![Diagram of student, transcript, and course entities]

The **grade** is functionally dependent on the entire primary key. The **course_name** attribute depends only on the **course_id**. To meet 2NF, move **course_name** to a separate entity.
Other Normal Forms

Additional normal forms, each building on the previous one, have been defined by relational database theorists.

- Boyce-Codd Normal Form (BCNF).
- Fourth Normal Form (4NF).
- Fifth Normal Form (5NF).
- Domain-Key Normal Form (DKNF).
- Sixth Normal Form (6NF).

- It is not necessarily desirable to apply these to every database.
  - Some are limited only to very particular types of tables.
  - Some are not always achievable.

Normalization results in a proliferation of tables and elimination of redundant columns.

- Sometimes this can lead to inconvenient complexity and poor performance.

Denormalization is the deliberate and judicious violation of normalization rules, usually to provide convenient access and higher performance.

- Denormalization results in redundant storage of an attribute in multiple tables.
- This often occurs, for example, in data warehouses.
The `advisor_name` attribute in the student table is dependent on the non-key `advisor_ssn` attribute.

Place the `student` table in 3NF by moving the `advisor_name` attribute to another entity.
Applications for Relational Databases

Perhaps the most familiar category of RDB is the on-line transaction processing (OLTP) database.

- An OLTP database supports large volumes of concurrent data manipulation language (DML) access during normal operation.
- Order processing, human resources, and web commerce systems are some typical OLTP applications.
- OLTP databases are usually fully normalized, in order to maintain the "ACID" properties of transactions:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

A data warehouse is designed and optimized for queries and analysis rather than data manipulation.

- Data warehouses frequently make use of denormalization and redundant copies of attribute values.
- Data warehouse tables are often read-only, to prevent any possibility of update anomalies.
- The data warehouse comprises the long-term historical record of database activity, usually derived from OLTP systems.
  - Part of a data warehouse design includes the procedures for extraction, transformation, and loading (ETL) of production data.
  - The data warehouse supports on-line analytical processing (OLAP).

Other special-purpose applications include spatial and temporal databases.
Normalization prevents insert, update, and deletion anomalies in an OLTP database. Normally, a data warehouse is non-volatile — that is, non-updatable, once the data is loaded. This allows the data warehouse designer to violate a number of classic database design rules in order to improve performance or facilitate particular kinds of analysis.

For example, Materialized Views (formerly known as snapshot tables) physically replicate data that's present in existing fact tables in order to provide more convenient query capabilities.

The transformation portion of a data warehouse's ETL solution may include aggregation and derivation of values. For example, to facilitate weekly sales reporting, individual sale attributes may be aggregated together to weekly granularity, and stored. Or, counts of inventory items may be pre-calculated and stored. In an OLTP database, such storage of values which are readily derivable from other values (which may change over time) can result in inconsistencies and faulty analysis. In read-only data warehouse tables, this is not a problem.

Because data warehouses accumulated ever-growing amounts of historical data, and because they frequently store multiple copies of data values, they tend to have very high storage, memory, and analytical processing requirements.
LABS

For this lab, we will describe requirements for two database applications: a Human Resources (HR) and an Order Entry (OE) system. For each, based on the description, we need to:

1. Identify the core entities and their important attributes.
2. Determine the cardinality (one-to-one, one-to-many, many-to-many) of the associations between entities.
3. Develop a logical model for the database:
   a. Determine table and column names.
   b. Determine primary key columns.
   c. Normalize, to 3NF, where possible.
   d. Identify foreign key references between tables.

Where possible, you can do this exercise in groups with other students. It is not necessary to produce a complete logical or physical model for both systems — just analyze and design the core tables.

(Solution: There is no single "correct" outcome for this exercise. After you are satisfied with your logical schema (or you run out of time), locate the Sample Schemas book in the Oracle Database online Documentation Library. Review the rationale, diagrams, and schema scripts for the Sample Schemas that Oracle provides. Compare them to your own design for further ideas on database analysis and design.)
HR:
Each employee is assigned an ID number. We need to record their name, email, job code, salary (some employees also earn commission), and manager ID.

Positions in the company are predefined, with a job code, job title, and a minimum and maximum salary range for each. As employees advance (or otherwise change positions), we want to record the employment duration on that job, the job identification number, and the department.

The company has presence in several regions, and has several locations in which warehouses are located. There are many departments (Administration, Marketing, Purchasing, Human Resources, Shipping, etc.). Each employee is assigned to a department, and each department has a unique department ID. Each department is associated with one location, which has a full address that includes the street name, postal code, city, state or province, and the country code.

For each of the various locations associated with warehouses and departments, we record the country name, currency symbol, currency name, and the geographical region.

OE:
For our products, such as computer hardware and software, music, clothing, and tools, we store information such as product ID, product category, order entries, the shipping weight group, the warranty period, the supplier, the availability, a list price, a minimum retail price, and a URL for manufacturer information. Inventory information includes the warehouse where the product is available and the quantity on hand. Each warehouse has a warehouse ID number, name, facility description, and location ID.

Each customer has an ID number. Customer information includes customer name, street name, city or province, country, phone numbers (up to five phone numbers per customer), and zip code. Some customers place orders online, so we store their email addresses. Customers have a credit limit. Some customers have an Account Manager at our company.

When a customer places an order, we record the date of the order, how it was placed, the current status, shipping mode, total amount, and the Sales Representative who served the customer. The Sales Representative may or may not be the same person as the customer's Account Manager. If an order is placed online, no sales rep is involved. Of course the order information includes the number of items ordered, the unit price, and the products ordered.
OBJECTIVES

* Establish goals in SQL tuning.
* Describe how Oracle's SQL Optimizer works.
* Identify which SQL to tune.
* Use hints to improve SQL performance.
* Select access paths for better performance.
* Improve join performance with hints.
* Take advantage of execution plan stability.
**Tuning Goals**

SQL performance tuning can involve many facets:
- Tuning SQL statements.
- Tuning the instance.
- Tuning the OS.

Some basic steps in the SQL tuning process include:
- Designing efficient SQL during application development.
- Identifying poorly-performing SQL.
- Correcting the execution plan for optimal performance.

What are some tuning goals that will lead to better application response time and lower resource usage?
- Reduce the workload.
  - Optimize your SQL so the database does less work in executing the SQL.
- Balance the workload.
  - Schedule intensive SQL during non-peak hours.
- Parallelize the workload.
  - Split the SQL into smaller jobs, executed in parallel.
Performance tuning is a very large topic which can take a long time to master. Performance tuning can be broken down into three major areas:

<table>
<thead>
<tr>
<th>Tuning Area</th>
<th>Average Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Tuning</td>
<td>75%</td>
</tr>
<tr>
<td>Instance Tuning</td>
<td>20%</td>
</tr>
<tr>
<td>OS Tuning</td>
<td>5%</td>
</tr>
</tbody>
</table>

Focusing on tuning SQL statements provides the biggest benefit. SQL statements should be tuned as the application is designed, but this can also be performed after the fact. This chapter focuses on SQL statement tuning.

A large part of tuning SQL statements focuses on reducing the workload, or doing the job with fewer resources. Examples of reducing the workload include:

- Reading fewer disk blocks in processing the SQL statement by employing or suppressing indexes.
- Utilizing a more efficient join method.
- Employing bind variables to reduce parsing.

**Note:**
When timing SQL queries, execute the query a couple of times before obtaining timing information to ensure the disk blocks are in the buffer cache.
THE OPTIMIZER

The Optimizer's job is to determine the most efficient method to execute a given SQL statement.

- In Oracle10g, the Optimizer does a pretty good job most of the time.
- The Optimizer is not always correct.
- The developer may know more about the data than the Optimizer does.

Earlier versions of Oracle used a Rule-Based Optimizer (RBO), which makes decisions based on a set of rules.

- The RBO is deprecated in Oracle10g.

The Cost-Based Optimizer (CBO) uses information about the data to make informed decisions.

- Each candidate execution plan is given a total cost and the execution plan with the lowest cost is chosen.
- The CBO needs statistics on the tables and indexes involved in the query.
The CBO relies on many different types of statistics in order to make the best-informed decisions.

Table statistics include:

- Number of rows
- Number of blocks
- Number of empty blocks
- Average row length
- Number of distinct values in a column
- Density of values in a column
- Number of NULL values in a column
- Histograms showing data distribution

Index statistics include:

- Height of the index tree
- Number of leaf blocks in the tree
- Number of distinct keys
- Average leaf blocks per key
- Clustering factor

System performance statistics include:

- I/O performance
- CPU performance
- CPU utilization
Optimizer Statistics

The CBO needs accurate statistics to make accurate decisions.

- Many SQL performance problems can be fixed simply by generating accurate statistics.

- Optimizer statistics are stored in the Data Dictionary.
  - `DBA_TABLES`
  - `DBA_TAB_COLUMNS`
  - `DBA_INDEXES`

- When generating statistics, Oracle can use all of the tables' rows (COMPUTE) or a representative sample (ESTIMATE).
  - Computing statistics can require significant resources — all rows of a table must be read just to generate the statistics, for example.
  - In Oracle10g, `ESTIMATE` is as good as `COMPUTE` in most cases.

- Oracle10g automates statistics gathering.
  - In previous versions, the DBA usually set up periodic jobs to regenerate statistics.
  - Oracle10g has a built-in Scheduler, with a predefined job to periodically update stats.

- Since the CBO can't create effective plans without good statistics, Oracle can dynamically sample data at SQL execution time if statistics are missing.
  - The `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter controls how it does this.

- The `DBMS_STATS` package allows you to manage Optimizer statistics.
Refresh the statistics for the HR schema:

```sql
SQL> EXEC DBMS_STATS.GATHER_SCHEMA_STATS('HR');
```

Set specific statistics for a table:

```sql
BEGIN
  DBMS_STATS.SET_TABLE_STATS(ownname => 'HR', tabname =>'MYTAB',
  numrows => 100, numblks => 10);
END;
/
```

View some of the statistics in the Data Dictionary:

```sql
SQL> SELECT num_rows, blocks FROM user_tables
WHERE table_name = 'MYTAB';
```

Regenerate statistics for a specific table:

```sql
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'MYTAB');
```
Identifying SQL to Tune

Oracle10g includes the Automatic Database Diagnostic Monitor (ADDM), a knowledge expert which can show the SQL statements that need tuning.

Oracle10g also include the Automatic Workload Repository (AWR), which automatically collects performance metrics every hour, including SQL statistics.

- ADDM and AWR reports use these statistics.

Prior to Oracle10g, the Oracle Statspack utility served a function similar to the AWR.

- It can be set up to collect performance statistics at regular intervals.

You can find SQL execution statistics in the dynamic performance view V$SQL.

- SQL statements sorted by disk reads per execution:

  ```sql
  SELECT sql_text,
         disk_reads/executions AS reads_per_exec
  FROM v$sql ORDER BY 2 DESC;
  ```

- SQL statements sorted by buffer gets per execution:

  ```sql
  SELECT sql_text,
         buffer_gets/executions AS gets_per_exec
  FROM v$sql ORDER BY 2 DESC;
  ```

In the Oracle10g Enterprise Manager, the Top SQL section helps you drill down into V$SQL.
ADDM was introduced in Oracle10g and is a great way of discovering the SQL statements that need tuning. This sample output from ADDM denotes a specific SQL statement (id 6jm6k9uwu6cp) that could benefit from tuning:

**FINDING 1: 100% impact (3556 seconds)**

SQL statements consuming significant database time were found.

**RECOMMENDATION 1: SQL Tuning, 100% benefit (3556 seconds)**

ACTION: Run SQL Tuning Advisor on the SQL statement with SQL_ID "6jm6k9uw8u6cp".

RELEVANT OBJECT: SQL statement with SQL_ID 6jm6k9uw8u6cp and PLAN_HASH 4163664868

```sql
select count(*) from all_objects,all_objects
```

Enterprise Manager contains a nice drill down into V$SQL called *Top SQL*. This feature lets you get a quick, graphical look at the SQL statements performing the most work in your instance. You can see an example of Top SQL in the figure below:

Clicking the **SQL ID** link for a specific SQL statement will let you drill down even further.
Optimizer Hints

* Even with the best statistics, the Optimizer may still derive a less-than-optimal execution plan.

* Hints direct the Optimizer towards a specific course of action.

* A hint is a comment with a plus sign in the SQL statement.

```
SELECT /*+ hint */ column_list FROM ...;
SELECT /*+ hint1 hint2 */ column_list FROM ...;
```

- The plus sign must follow the asterisk with no spaces.
- If a table is aliased in the FROM clause, you must use the alias in the hint, otherwise the hint is ignored.
- If the hint is misspelled or used improperly, the Optimizer will ignore the hint without any indication of errors.

* Performance-tuning hints should be used sparingly, as changes in Oracle versions may cause queries to execute differently.

- Oracle recommends using the Advisor Framework and Plan Management.
- However, hints are still used to toggle certain features and behaviors.

* Optimizer hints can be categorized by their function.

- Optimizer Goal hints
- Access Path hints
- Join hints
- Miscellaneous hints
Optimizer hints fall into the following classifications:

- *Single table hints* give direction on how to handle just one table.

- *Multi-table hints* give direction on how to handle more than one table. Join hints fall into this category.

- *Query block hints* give direction on how to handle the block of SQL code. Performing query rewrite and pushing a subquery into the main query are examples.

- *Statement hints* give direction on how to handle the entire statement. The optimizer goal hints are classified as statement hints.
Optimizer Goal Hints

- The Optimizer can either return results as the rows become available or all at once.

- The `ALL_ROWS` hint instructs the Optimizer to return all rows of the query as fast as possible.

```sql
SELECT /*+ ALL_ROWS */ * FROM hr.employees;
```

- `ALL_ROWS` minimizes the overall resource consumption.

- This is typically a good Optimizer goal for most applications.

- The `FIRST_ROWS` hint instructs the Optimizer to return the first few rows, then go back to work on the rest of the query.

```sql
SELECT /*+ FIRST_ROWS(10) */ * FROM hr.employees;
```

- This attempts to return the first $n$ rows as efficiently as possible.

- `FIRST_ROWS` is great for Oracle Forms applications.
The **RULE** hint still exists in Oracle10g to force the Optimizer to use the old Rule-Based Optimizer. This hint should be avoided in Oracle10g, as it is no longer supported. In earlier versions of Oracle (pre-Oracle9i), where the Cost-Based Optimizer was not nearly as robust as it is today, you may wish to use the **RULE** hint.

The **ALL_ROWS** and **FIRST_ROWS(n)** hints are used to override a session or instance's setting for the **OPTIMIZER_MODE** parameter. If you feel the need to code all of your session's statements with a specific optimizer mode hint, you may find it more beneficial to alter the session similar to the following:

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS(100);
```
Access Path Hints

- An access path is the route Oracle takes to obtain the rows of data from a table.
- Some common access paths are to read the entire table (a full table scan) or to utilize an index (an index scan).
- A full table scan reads all rows of the table, up to the High Water Mark.

hints2.sql

```sql
SELECT /*+ FULL (employee) */ employee_id, last_name
FROM hr.employees WHERE employee_id = 201;
```

- An index scan uses a specific index.

```sql
SELECT /*+ INDEX (d dept_loc_idx) */ department_name
FROM hr.departments d WHERE location_id = 2400;
```

- A no index hint suppresses a specific index from being used.

```sql
SELECT /*+ NO_INDEX (d dept_loc_idx) */
    department_name
FROM hr.departments d WHERE location_id = 1700;
```

- If an index is specified in the hint, the index is ignored, but other indexes can be considered.
- If the hint does not denote any indexes, then no indexes are considered.
Access path hints:

- CLUSTER
- FULL
- HASH
- INDEX
- INDEX_ASC
- INDEX_DESC
- INDEX_COMBINE
- INDEX_JOIN
- INDEX_FFS
- INDEX_SS
- INDEX_SS_ASC
- INDEX_SS_DESC
- NO_INDEX
- NO_INDEX_FFS
- NO_INDEX_SS
JOIN HINTS

Hints can suggest ways for Oracle to process a join.

Join order joins tables in the order they appear in the FROM clause.

```
join3.sql
SELECT /*+ ORDERED */
  first_name, last_name, department_name
FROM hr.employees JOIN hr.departments
  USING (department_id);
```

Normally, the Optimizer will consider all possible join orders.

The Hash Join Method builds a hash table of the key values from one (usually smaller) table, then scans the larger table to find matches.

```
SELECT /*+ USE_HASH(e d) */
  first_name, last_name, department_name
FROM hr.employees JOIN hr.departments
  USING (department_id);
```

A Merge Sort Join Method sorts both tables on the join key, then merges them together.

```
SELECT /*+ USE_MERGE(e d) */
  first_name, last_name, department_name
FROM hr.employees JOIN hr.departments
  USING (department_id);
```

Nested Loops Join Method — For each row in one table (the driving table), access each row of the other table.

```
SELECT /*+ USE_NL(e d) */
  first_name, last_name, department_name
FROM hr.employees JOIN hr.departments
  USING (department_id);
```
Join hints:

- ORDERED
- LEADING
- USE_HASH
- USE_MERGE
- USE_NL
- USE_NL_WITH_INDEX
- NO_USE_HASH
- NO_USE_MERGE
- NO_USE_NL

There are three different join algorithms to process join operations in the Oracle database. The Nested Loop Join, the Sort Merge Join, and the Hash Join. When tuning queries that involve join operations, you may want to try each of the three join algorithms to see which one works best for your query.
Additional Hints

If the instance is configured to run multiple parallel query processes, a Parallel Execution Hint can affect a query's execution.

- PARALLEL — Overrides the degree of parallelism.
- NO_PARALLEL — Does not execute as a parallel query.

Query Transformation Hints affect how views are queried.

- MERGE — Merges views into the query.
- REWRITE — Allows query rewrite for materialized views.

The Optimizer provides a variety of other hints.

- APPEND — Uses direct path insert.
- CACHE — Places blocks in the most recently used end of LRU list of the buffer cache for a full table scan.
- CURSOR_SHARING_EXACT — Switches off cursor sharing.
- DRIVING_SITE — Changes the site which runs a distributed query.
Parallel hints:

- PARALLEL_INDEX
- NO_PARALLEL_INDEX
- PQ_DISTRIBUTE

Query transformation hints:

- FACT
- NO_FACT
- NO_MERGE
- NO_EXPAND
- USE_CONCAT
- NO_REWRITE
- UNNEST
- NO_UNNEST
- STAR_TRANSFORMATION
- NO_STAR_TRANSFORMATION
- NO_QUERY_TRANSFORMATION

Miscellaneous hints:

- NO_APPEND
- NO_CACHE
- PUSH_PRED
- NO_PUSH_PRED
- PUSH_SUBQ
- NO_PUSH_SUBQ
- PX_JOIN_FILTER
- NO_PX_JOIN_FILTER
- NO_XML_QUERY_REWRITE
- QB_NAME
- MODEL_MIN_ANALYSIS
- DYNAMIC_SAMPLING
SQL Plan Management

- The performance of the same plan can change over time due to changes in statistics, parameters, newer Oracle versions, schema changes, and so on.

- SQL plan management keeps track of execution plans and their performance over time.
  - You can build a collection of proven-successful plans, called a SQL plan baseline.

- If the OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES system parameter is changed from its default false to true, Oracle automatically captures and maintains SQL execution plans in the database.
  - The first plan for a statement is marked as accepted.
  - All subsequent plans for that statement are added to its plan history.
  - Plans that don't cause performance to worsen are marked accepted.
    - Accepted plans comprise the plan baseline.

- You can manually generate a plan baseline by loading existing execution plans from SQL tuning sets, AWR (Automatic Workload Repository) snapshots, or even the Shared SQL area.

- When the Optimizer parses a statement, it will compare its best-cost plan with those in the baseline and choose the lowest-cost of these.
  - If the baseline contains the same plan as the Optimizer's, it will use that.
SQL Plan Management replaces the now-deprecated plan stability via Stored Outlines. Unlike stored outlines, SQL baselines can automatically evolve over time.

The DBMS_SPM supplied package provides procedures to migrate stored outlines to the SQL Plan Management base. See Oracle Database Performance Tuning Guide - Using SQL Plan Management for details.
Labs

1. Create a large table named **TEST_TABLE** by doing a **CREATE TABLE AS SELECT**. For this, select the **OBJECT_ID**, **OWNER**, **OBJECT_NAME**, and **OBJECT_TYPE** from **ALL_OBJECTS**. Then, to increase the number of rows, use this **INSERT** statement:

```sql
SQL> INSERT INTO test_table
2   SELECT object_id+100000, owner, object_name, object_type
3   FROM test_table;
```

Repeat the **INSERT** command until you have more than 5 million rows of data in the table. (Solution: `create_large_table.sql`)

2. Ensure statistics for your schema are up to date, by manually invoking the **GATHER_SCHEMA_STATS** procedure from **DBMS_STATS**. (Solution: `gather_stats.sql`)

3. In SQL*Plus, set **TIMING** on so that you can see how long a query takes to execute. Turn off the **PAUSE** setting also. Then from the **USER_OBJECTS** table, determine the **OBJECT_ID** of the **HR.EMPLOYEES** table:

```sql
SQL> SELECT owner, object_id FROM all_objects
2   WHERE object_name= 'EMPLOYEES' ;
```

Using that **OBJECT_ID**, query for the corresponding row in your large test table. Repeat the query a few times, and note the elapsed time. (Solution: `time_query.sql`)

4. Create an index on the **OBJECT_ID** column of **TEST_TABLE**. Run the previous query for the same **OBJECT_ID** again; repeat the query a few times, and note the elapsed time. Did the index improve query performance? (Solution: `time_wth_index.sql`)

5. Repeat the query, this time using a hint to suppress the use of the newly-created index. Repeat the query a few times, noting the elapsed time. Did the hint have any effect on query performance? (Solution: `time_wth_hint.sql`)
6 Write a query that joins TEST_TABLE to itself (using table aliases), joining on the OBJECT_ID column. Have the query do just a COUNT(*). Note the execution time. Now, repeat the query, using each of the three join hints (nested loop, sort merge, hash join), noting the elapsed time for each. Which hint gave the best performance for this example?
(Solution: time_join_hints.sql)

7 Query the Data Dictionary to determine the number of rows and blocks of the test table. Using the DBMS_STATS.SET_TABLE_STATS procedure, modify the TEST_TABLE's statistics to fool the Optimizer into thinking that the table contains only 1 row in 1 block. Query the Data Dictionary to verify the statistics on the table have changed.
(Solution: modify_stats.sql)