

Oracle Database 10g: SQL Fundamentals I

Student Guide • Volume 1

D17108GC30

Edition 3.0

January 2009

D57870

ORACLE®

Authors

Salome Clement
 Chaitanya Koratamaddi
 Nancy Greenberg

Technical Contributors**and Reviewers**

Wayne Abbott
 Christian Bauwens
 Claire Bennett
 Perry Benson
 Brian Boxx
 Zarko Cesljas
 Dairy Chan
 Laszlo Czinkoczki
 Joel Goodman
 Matthew Gregory
 Sushma Jagannath
 Yash Jain
 Angelika Krupp
 Isabelle Marchand
 Malika Marghadi
 Valli Pataballa
 Narayanan Radhakrishnan
 Bryan Roberts
 Helen Robertson
 Lata Shivaprasad
 John Soltani
 James Spiller
 Priya Vennapusa

Editors

Arijit Ghosh
 Raj Kumar

Graphic Designer

Rajiv Chandrabhanu

Publisher

Giri Venugopal

Copyright © 2009, Oracle. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

Preface

I Introduction

Lesson Objectives	I-2
Goals of the Course	I-3
Oracle10g	I-4
Oracle Database 10g	I-6
Oracle Application Server 10g	I-7
Oracle Enterprise Manager 10g Grid Control	I-8
Relational and Object Relational Database Management Systems	I-9
Oracle Internet Platform	I-10
System Development Life Cycle	I-11
Data Storage on Different Media	I-13
Relational Database Concept	I-14
Definition of a Relational Database	I-15
Data Models	I-16
Entity Relationship Model	I-17
Entity Relationship Modeling Conventions	I-19
Relating Multiple Tables	I-21
Relational Database Terminology	I-23
Relational Database Properties	I-25
Communicating with an RDBMS Using SQL	I-26
Oracle's Relational Database Management System	I-27
SQL Statements	I-28
Tables Used in the Course	I-29
Summary	I-30

1 Retrieving Data Using the SQL SELECT Statement

Statement Objectives	1-2
Capabilities of SQL SELECT Statements	1-3
Basic SELECT Statement	1-4
Selecting All Columns	1-5
Selecting Specific Columns	1-6
Writing SQL Statements	1-7
Column Heading Defaults	1-8

Arithmetic Expressions	1-9
Using Arithmetic Operators	1-10
Operator Precedence	1-11
Defining a Null Value	1-12
Null Values in Arithmetic Expressions	1-13
Defining a Column Alias	1-14
Using Column Aliases	1-15
Concatenation Operator	1-16
Literal Character Strings	1-17
Using Literal Character Strings	1-18
Alternative Quote (q) Operator	1-19
Duplicate Rows	1-20
Development Environments for SQL	1-21
What Is Oracle SQL Developer?	1-22
Oracle SQL Developer Interface	1-23
Creating a Database Connection	1-24
Browsing Database Objects	1-27
Using the SQL Worksheet	1-28
Executing SQL Statements	1-31
Formatting the SQL Code	1-32
Saving SQL Statements	1-33
Running Script Files	1-34
Displaying the Table Structure	1-35
Using the DESCRIBE Command	1-36
Summary	1-37
Practice 1: Overview	1-38

2 Restricting and Sorting

Data Objectives	2-2
Limiting Rows Using a Selection	2-3
Limiting the Rows That Are Selected	2-4
Using the WHERE Clause	2-5
Character Strings and Dates	2-6
Comparison Conditions	2-7
Using Comparison Conditions	2-8
Using the BETWEEN Condition	2-9
Using the IN Condition	2-10
Using the LIKE Condition	2-11
Using the NULL Conditions	2-13
Logical Conditions	2-14

Using the AND Operator 2-15
Using the OR Operator 2-16
Using the NOT Operator 2-17
Rules of Precedence 2-18
Using the ORDER BY Clause 2-20
Sorting 2-21
Substitution Variables 2-22
Using the & Substitution Variable 2-24
Character and Date Values with Substitution Variables 2- 26
Specifying Column Names, Expressions, and Text 2-27
Using the && Substitution Variable 2-28
Using the DEFINE Command 2-29
Using the VERIFY Command 2-30
Summary 2-31
Practice 2: Overview 2-32

3 Using Single-Row Functions to Customize

Output Objectives 3-2
SQL Functions 3-3
Two Types of SQL Functions 3-4 Single-
Row Functions 3-5 Character Functions
3-7 Case-Manipulation Functions 3-9
Using Case-Manipulation Functions 3-10
Character-Manipulation Functions 3-11

Using the Character-Manipulation Functions 3-12
Number Functions 3-13
Using the ROUND Function 3-14
Using the TRUNC Function 3-15
Using the MOD Function 3-16
Working with Dates 3-17
Arithmetic with Dates 3-20
Using Arithmetic Operators with Dates 3-21
Date Functions 3-22
Using Date Functions 3-23
Practice 3: Overview of Part 1 3-25
Conversion Functions 3-26
Implicit Data Type Conversion 3-27
Explicit Data Type Conversion 3-29
Using the TO_CHAR Function with Dates 3-32

Elements of the Date Format Model 3-33 Using the
TO_CHAR Function with Dates 3-37 Using the
TO_CHAR Function with Numbers 3-38 Using the
TO_NUMBER and TO_DATE Functions 3-41
RR Date Format 3-43
RR Date Format: Example 3-44
Nesting Functions 3-45 General
Functions 3-47
NVL Function 3-48
Using the NVL Function 3-49 Using
the NVL2 Function 3-50 Using the
NULLIF Function 3-51 Using the
COALESCE Function 3-52
Conditional Expressions 3-54 CASE
Expression 3-55
Using the CASE Expression 3-56
DECODE Function 3-57
Using the DECODE Function 3-58
Summary 3-60
Practice 3: Overview of Part 2 3-61

4 Reporting Aggregated Data Using the Group Functions

Objectives 4-2
What Are Group Functions? 4-3
Types of Group Functions 4-4
Group Functions: Syntax 4-5
Using the AVG and SUM Functions 4-6
Using the MIN and MAX Functions 4-7
Using the COUNT Function 4-8 Using
the DISTINCT Keyword 4-9 Group
Functions and Null Values 4-10
Creating Groups of Data 4-11
Creating Groups of Data: GROUP BY Clause Syntax 4-12
Using the GROUP BY Clause 4-13
Grouping by More Than One Column 4-15
Using the GROUP BY Clause on Multiple Columns 4-16
Illegal Queries Using Group Functions 4-17
Restricting Group Results 4-19
Restricting Group Results with the HAVING Clause 4-20

Using the HAVING Clause	4-21
Nesting Group Functions	4-23
Summary	4-24
Practice 4: Overview	4-25

5 Displaying Data from Multiple Tables

Objectives	5-2
Obtaining Data from Multiple Tables	5-3
Types of Joins	5-4
Joining Tables Using SQL:1999 Syntax	5-5
Creating Natural Joins	5-6
Retrieving Records with Natural Joins	5-7
Creating Joins with the USING Clause	5-8
Joining Column Names	5-9
Retrieving Records with the USING Clause	5-10
Qualifying Ambiguous Column Names	5-11
Using Table Aliases	5-12
Creating Joins with the ON Clause	5-13
Retrieving Records with the ON Clause	5-14
Self-Joins Using the ON Clause	5-15
Applying Additional Conditions to a Join	5-17
Creating Three-Way Joins with the ON Clause	5-18
Nonequijoins	5-19
Retrieving Records with Nonequijoins	5-20
Outer Joins	5-21
INNER Versus OUTER Joins	5-22
LEFT OUTER JOIN	5-23
RIGHT OUTER JOIN	5-24
FULL OUTER JOIN	5-25
Cartesian Products	5-26
Generating a Cartesian Product	5-27
Creating Cross Joins	5-28
Summary	5-29
Practice 5: Overview	5-30

6 Using Subqueries to Solve Queries

Objectives	6-2
Using a Subquery to Solve a Problem	6-3
Subquery Syntax	6-4
Using a Subquery	6-5

Guidelines for Using Subqueries	6-6
Types of Subqueries	6-7
Single-Row Subqueries	6-8
Executing Single-Row Subqueries	6-9
Using Group Functions in a Subquery	6-10
The <code>HAVING</code> Clause with Subqueries	6-11
What Is Wrong with This Statement?	6-12
Will This Statement Return Rows?	6-13
Multiple-Row Subqueries	6-14
Using the <code>ANY</code> Operator in Multiple-Row Subqueries	6-15
Using the <code>ALL</code> Operator in Multiple-Row Subqueries	6-16
Null Values in a Subquery	6-17
Summary	6-19
Practice 6: Overview	6-20

7 Using the Set Operators

Objectives	7-2
Set Operators	7-3
Tables Used in This Lesson	7-4
<code>UNION</code> Operator	7-8
Using the <code>UNION</code> Operator	7-9
<code>UNION ALL</code> Operator	7-11
Using the <code>UNION ALL</code> Operator	7-12
<code>INTERSECT</code> Operator	7-13
Using the <code>INTERSECT</code> Operator	7-14
<code>MINUS</code> Operator	7-15
Set Operator Guidelines	7-17
The Oracle Server and Set Operators	7-18
Matching the <code>SELECT</code> Statements	7-19
Matching the <code>SELECT</code> Statement: Example	7-20
Controlling the Order of Rows	7-21
Summary	7-22
Practice 7: Overview	7-23

8 Manipulating Data

Objectives	8-2
Data Manipulation Language	8-3
Adding a New Row to a Table	8-4
<code>INSERT</code> Statement Syntax	8-5
Inserting New Rows	8-6

Inserting Rows with Null Values	8-7
Inserting Special Values	8-8
Inserting Specific Date Values	8-9
Creating a Script	8-10
Copying Rows from Another Table	8-11
Changing Data in a Table	8-12
UPDATE Statement Syntax	8-13
Updating Rows in a Table	8-14
Updating Two Columns with a Subquery	8-15
Updating Rows Based on Another Table	8-16
Removing a Row from a Table	8-17
DELETE Statement	8-18
Deleting Rows from a Table	8-19
Deleting Rows Based on Another Table	8-20
TRUNCATE Statement	8-21
Using a Subquery in an INSERT Statement	8-22
Database Transactions	8-24
Advantages of COMMIT and ROLLBACK Statements	8-26
Controlling Transactions	8-27
Rolling Back Changes to a Marker	8-28
Implicit Transaction Processing	8-29
State of the Data Before COMMIT or ROLLBACK	8-31
State of the Data After COMMIT	8-32
Committing Data	8-33
State of the Data After ROLLBACK	8-34
Statement-Level Rollback	8-36
Read Consistency	8-37
Implementation of Read Consistency	8-38
Summary	8-39
Practice 8: Overview	8-40

9 Using DDL Statements to Create and Manage Tables

Objectives	9-2
Database Objects	9-3
Naming Rules	9-4
CREATE TABLE Statement	9-5
Referencing Another User's Tables	9-6
DEFAULT Option	9-7
Creating Tables	9-8
Data Types	9-9

Datetime Data Types	9-11
Including Constraints	9-17
Constraint Guidelines	9-18
Defining Constraints	9-19
NOT NULL Constraint	9-21
UNIQUE Constraint	9-22
PRIMARY KEY Constraint	9-24
FOREIGN KEY Constraint	9-25
FOREIGN KEY Constraint: Keywords	9-27
CHECK Constraint	9-28
CREATE TABLE: Example	9-29
Violating Constraints	9-30
Creating a Table by Using a Subquery	9-32
ALTER TABLE Statement	9-34
Dropping a Table	9-35
Summary	9-36
Practice 9: Overview	9-37

10 Creating Other Schema Objects

Objectives	10-2
Database Objects	10-3
What Is a View?	10-4
Advantages of Views	10-5
Simple Views and Complex Views	10-6
Creating a View	10-7
Retrieving Data from a View	10-10
Modifying a View	10-11
Creating a Complex View	10-12
Rules for Performing DML Operations on a View	10-13
Using the WITH CHECK OPTION Clause	10-16
Denying DML Operations	10-18
Removing a View	10-20
Practice 10: Overview of Part 1	10-21
Sequences	10-22
CREATE SEQUENCE Statement: Syntax	10-24
Creating a Sequence	10-25
NEXTVAL and CURRVAL Pseudocolumns	10-26
Using a Sequence	10-28
Caching Sequence Values	10-29
Modifying a Sequence	10-30

Guidelines for Modifying a Sequence 10-31
Indexes 10-33
How Are Indexes Created? 10-35
Creating an Index 10-36
Index Creation Guidelines 10-37
Removing an Index 10-38
Synonyms 10-39
Creating and Removing Synonyms 10-41
Summary 10-42
Practice 10: Overview of Part 2 10-43

11 Managing Objects with Data Dictionary Views

Objectives 11-2
The Data Dictionary 11-3 Data
Dictionary Structure 11-4
How to Use the Dictionary Views 11-6
USER_OBJECTS and ALL_OBJECTS Views 11-
7 USER_OBJECTS View 11-8
Table Information 11-9 Column
Information 11-10 Constraint
Information 11-12 View Information
11-15 Sequence Information 11-16
Synonym Information 11-18 Adding
Comments to a Table 11-19
Summary 11-20

Practice 11: Overview 11-21

A Practice Solutions

B Table Descriptions and Data

C Oracle Join Syntax

D Using SQL*Plus

E Using SQL Developer

Index

Additional Practices

Additional Practices: Table Descriptions and Data

Additional Practices: Solutions

Preface

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

Profile

Before You Begin This Course

Before you begin this course, you should be able to use a graphical user interface (GUI). The prerequisite is a familiarity with data processing concepts and techniques.

How This Course Is Organized

Oracle Database 10g: SQL Fundamentals I is an instructor-led course featuring lectures and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills that are introduced.

Related Publications

Oracle Publications

Title	Part Number
<i>Oracle® Database Reference 10g Release 2 (10.2)</i>	B14237-02
<i>Oracle® Database SQL Reference 10g Release 2 (10.2)</i>	B14200-02
<i>Oracle® Database Concepts 10g Release 2 (10.2)</i>	B14220-02
<i>Oracle® Database Application Developer's Guide - Fundamentals 10g Release 2 (10.2)</i>	B14251-01
<i>SQL*Plus® User's Guide and Reference</i>	B14357-01

Additional Publications

- System release bulletins
- Installation and user's guides
- *read.me* files
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

Typographic Conventions

The following two lists explain Oracle University typographical conventions for words that appear within regular text or within code samples.

1. Typographic Conventions for Words Within Regular Text

Convention	Object or Term	Example
Courier New	User input; commands; column, table, and schema names; functions; PL/SQL objects; paths	Use the SELECT command to view information stored in the LAST_NAME column of the EMPLOYEES table. Enter 300. Log in as scott
Initial cap	Triggers; user interface object names, such as button names	Assign a When-Validate-Item trigger to the ORD block. Click the Cancel button.
Italic	Titles of courses and manuals; emphasized words or phrases; placeholders or variables	For more information on the subject see <i>Oracle SQL Reference Manual</i> Do <i>not</i> save changes to the database. Enter <i>hostname</i> , where <i>hostname</i> is the host on which the password is to be changed.
Quotation marks	Lesson or module titles referenced within a course	This subject is covered in Lesson 3, “Working with Objects.”

Typographic Conventions (continued)

2. Typographic Conventions for Words Within Code Samples

Convention	Object or Term	Example
Uppercase	Commands, functions	SELECT employee_id FROM employees;
Lowercase, italic	Syntax variables	CREATE ROLE <i>role</i> ;
Initial cap	Forms triggers	Form module: ORD Trigger level: S _ITEM.QUANTITY item Trigger name: When-Validate-Item
Lowercase	Column names, table names, filenames, PL/SQL objects	OG_ACTIVATE_LAYER (OG_GET_LAYER ('prod_pie_layer')) . . . SELECT last_name FROM employees;
Bold	Text that must be entered by a user	CREATE USER scott IDENTIFIED BY tiger ;

I

Introduction

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Lesson Objectives

After completing this lesson, you should be able to do the following:

- List the features of Oracle10g
- Discuss the theoretical and physical aspects of a relational database
- Describe the Oracle implementation of RDBMS and ORDBMS
- Understand the goals of the course



Copyright © 2009, Oracle. All rights reserved.

Objectives

In this lesson, you gain an understanding of the relational database management system (RDBMS) and the object relational database management system (ORDBMS). You are also introduced to the following:

- SQL statements that are specific to Oracle
- SQL Developer, which is an environment used for executing SQL statements and for formatting and reporting purposes

Goals of the Course

After completing this course, you should be able to do the following:

- Identify the major structural components of Oracle Database 10g
- Retrieve row and column data from tables with the SELECT statement
- Create reports of sorted and restricted data
- Employ SQL functions to generate and retrieve customized data
- Run data manipulation language (DML) statements to update data in Oracle Database 10g
- Obtain metadata by querying the dictionary views

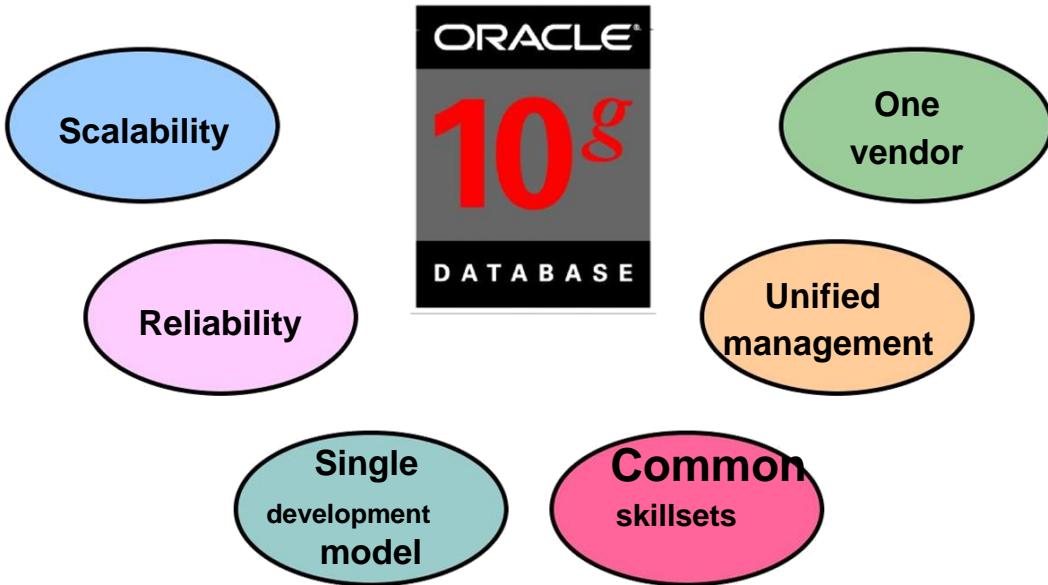


Copyright © 2009, Oracle. All rights reserved.

Goals of the Course

This course offers you an introduction to Oracle Database 10g database technology. In this class, you learn the basic concepts of relational databases and the powerful SQL programming language. This course provides the essential SQL skills that enable you to write queries against single and multiple tables, manipulate data in tables, create database objects, and query metadata.

Oracle10g



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle10g Features

The Oracle10g release offers a comprehensive high-performance infrastructure, including:

- Scalability from departments to enterprise e-business sites
- Robust, reliable, available, and secure architecture
- One development model; easy deployment options
- Leverage an organization's current skillset throughout the Oracle platform (including SQL, PL/SQL, Java, and XML)
- One management interface for all applications
- Industry-standard technologies; no proprietary lock-in

In addition to providing the benefits listed above, the Oracle10g release contains the database for the grid. Grid computing can dramatically lower the cost of computing, extend the availability of computing resources, and deliver higher productivity and quality.

The basic idea of grid computing is the notion of computing as a utility, analogous to the electric power grid or the telephone network. As a client of the grid, you do not care where your data is or where your computation is done. You want to have your computation done and to have your information delivered to you when you want it. From the server side, grid is about virtualization and provisioning. You pool all your resources together and provision these resources dynamically based on the needs of your business, thus achieving better resource utilization at the same time.

Oracle10g



ORACLE®

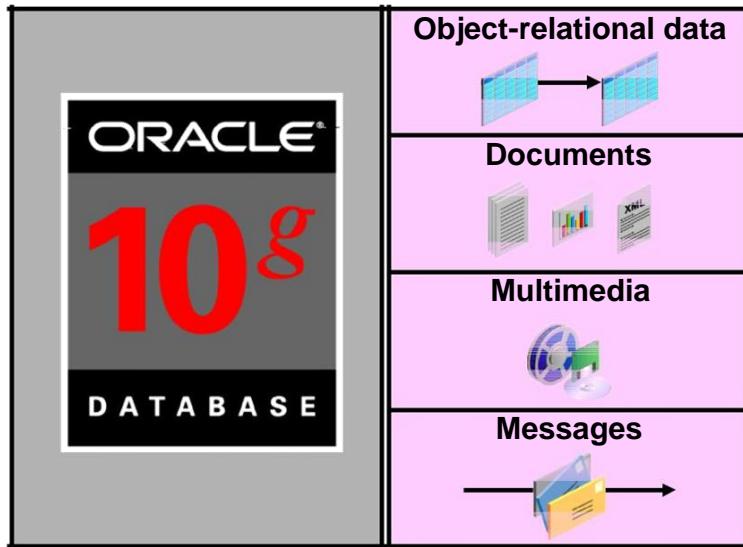
Copyright © 2009, Oracle. All rights reserved.

Oracle10g

The three grid-infrastructure products of the Oracle10g release are:

- Oracle Database 10g
- Oracle Application Server 10g
- Oracle Enterprise Manager 10g Grid Control

Oracle Database 10g



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 10g

Oracle Database 10g is designed to store and manage enterprise information. Oracle Database 10g cuts management costs and provides a high quality of service. Reduced configuration and management requirements and automatic SQL tuning have significantly reduced the cost of maintaining the environment.

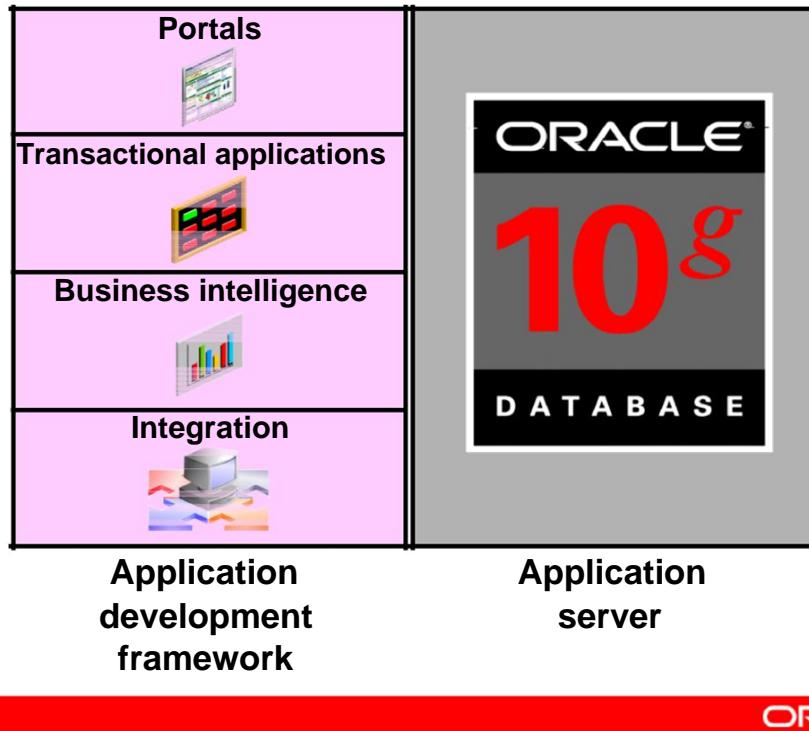
Oracle Database 10g contributes to the grid-infrastructure products of the Oracle 10g release. Grid computing is all about computing as a utility. If you are a client, you need not know where your data resides and which computer stores it. You should be able to request information or computation on your data and have it delivered to you.

Oracle Database 10g manages all your data. This is not just the object-relational data that you expect an enterprise database to manage. It can also be unstructured data such as:

- Spreadsheets
- Word documents
- PowerPoint presentations
- XML
- Multimedia data types like MP3, graphics, video, and more

The data does not even have to be in the database. Oracle Database 10g has services through which you can store metadata about information stored in file systems. You can use the database server to manage and serve information wherever it is located.

Oracle Application Server 10g



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Oracle Application Server 10g

Oracle Application Server 10g provides a complete infrastructure platform for developing and deploying enterprise applications, integrating many functions including a J2EE and Web services run-time environment, an enterprise portal, an enterprise integration broker, business intelligence, Web caching, and identity management services.

Oracle Application Server 10g adds new grid computing features, building on the success of Oracle9i Application Server, which has hundreds of customers running production enterprise applications. Oracle Application Server 10g is the only application server to include services for all the different server applications that you might want to run, including:

- Portals or Web sites
- Java transactional applications
- Business intelligence applications

It also provides integration among users, applications, and data throughout your organization.

Oracle Enterprise Manager 10g Grid Control

- Software provisioning
- Application service level monitoring



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Enterprise Manager 10g Grid Control

Oracle Enterprise Manager 10g Grid Control is the complete, integrated, central management console and underlying framework that automates administrative tasks across sets of systems in a grid environment. With Oracle Grid Control, you can group multiple hardware nodes, databases, application servers, and other targets into single logical entities. By executing jobs, enforcing standard policies, monitoring performance, and automating many other tasks across a group of targets instead of on many systems individually, Grid Control enables scaling with a growing grid.

Software Provisioning

With Grid Control, Oracle 10g automates installation, configuration, and cloning of Application Server 10g and Database 10g across multiples nodes. Oracle Enterprise Manager provides a common framework for software provisioning and management, enabling administrators to create, configure, deploy, and utilize new servers with new instances of the application server and database as they are needed.

Application Service Level Monitoring

Oracle Grid Control views the availability and performance of the grid infrastructure as a unified whole, as a user would experience it, rather than as isolated storage units, processing boxes, databases, and application servers.

Relational and Object Relational Database Management Systems

- Relational model and object relational model
- User-defined data types and objects
- Fully compatible with relational database
- Support of multimedia and large objects
- High-quality database server features



Copyright © 2009, Oracle. All rights reserved.

About the Oracle Server

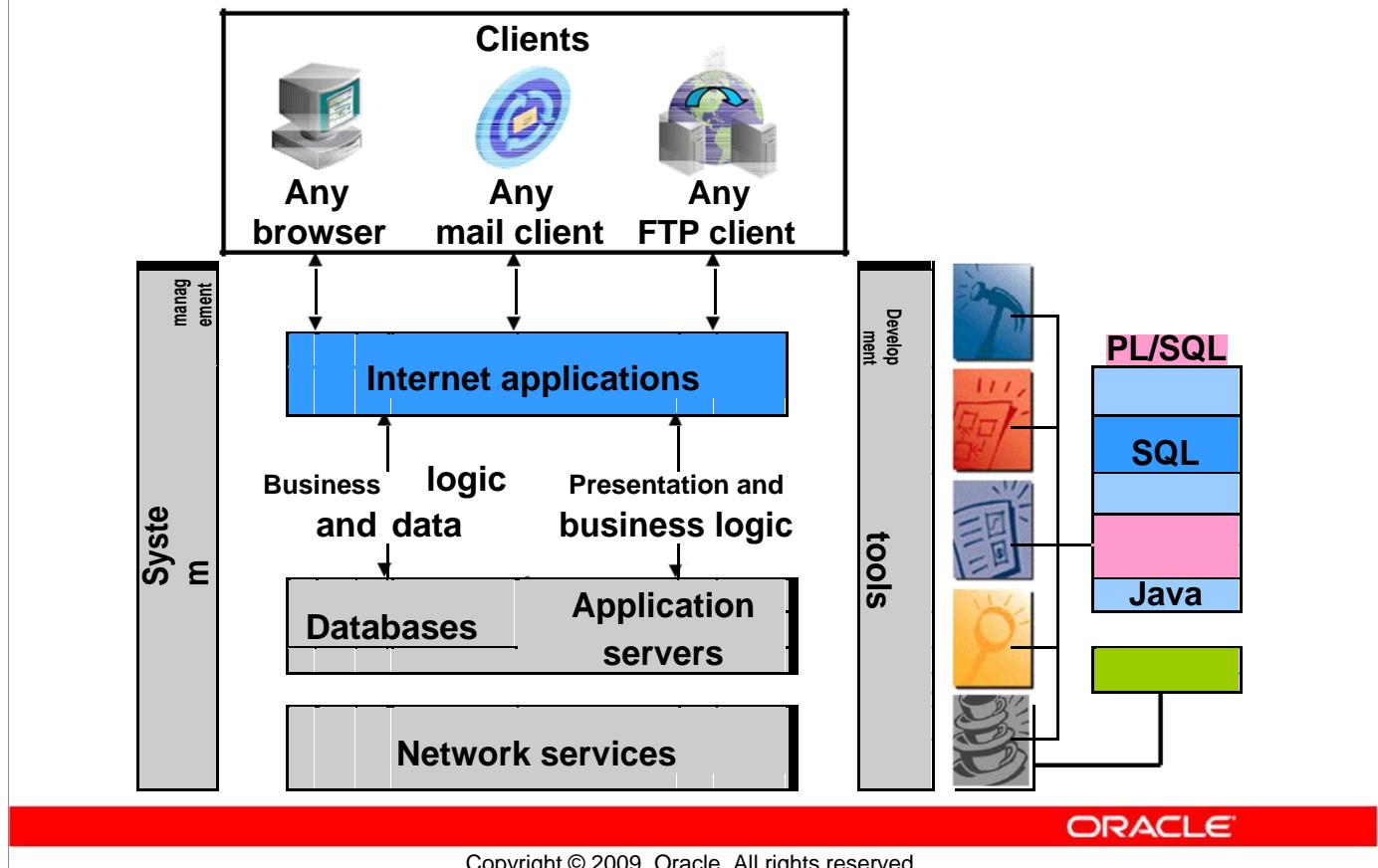
The Oracle server supports both the relational and object relational models.

The Oracle server extends the data-modeling capabilities to support an object relational database model that brings object-oriented programming, complex data types, complex business objects, and full compatibility with the relational world.

It includes several features for improved performance and functionality of online transaction processing (OLTP) applications, such as better sharing of run-time data structures, larger buffer caches, and deferrable constraints. Data warehouse applications benefit from enhancements such as parallel execution of insert, update, and delete operations; partitioning; and parallel-aware query optimization. Operating within the Network Computing Architecture (NCA) framework, the Oracle model supports client/server and Web-based applications that are distributed and multitiered.

For more information about the relational and object relational model, see the *Database Concepts* manual.

Oracle Internet Platform



Copyright © 2009, Oracle. All rights reserved.

Oracle Internet Platform

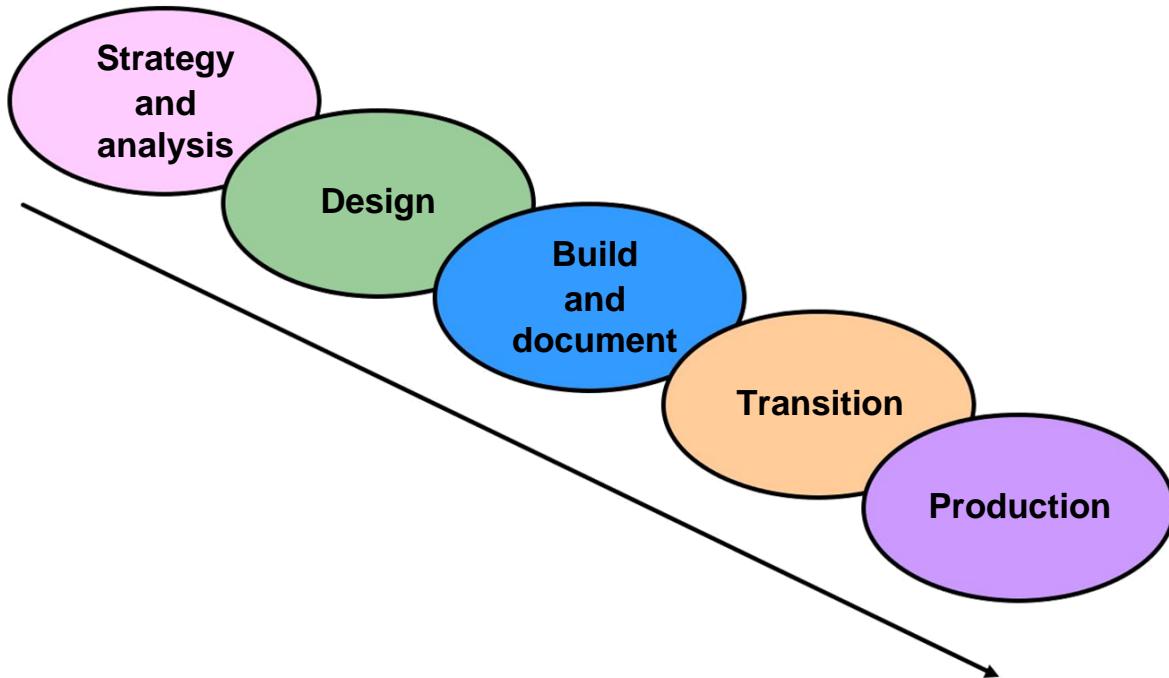
To develop an e-commerce application, you need a product that can store and manage the data, a product that can provide a run-time environment for your applications implementing business logic, and a product that can monitor and diagnose the application after it is integrated. The Oracle 10g products provide all the necessary components to develop your application.

Oracle offers a comprehensive high-performance Internet platform for e-commerce and data warehousing. The integrated Oracle Internet Platform includes everything needed to develop, deploy, and manage Internet applications, including these three core pieces:

- Browser-based clients to process presentation
- Application servers to execute business logic and serve presentation logic to browser-based clients
- Databases to execute database-intensive business logic and serve data

Oracle offers a wide variety of the most advanced graphical user interface (GUI)-driven development tools to build business applications, as well as a large suite of software applications for many areas of business and industry. Oracle Developer Suite includes tools to develop forms and reports and to build data warehouses. Stored procedures, functions, and packages can be written using SQL, PL/SQL, or Java.

System Development Life Cycle



ORACLE

Copyright © 2009, Oracle. All rights reserved.

System Development Life Cycle

From concept to production, you can develop a database by using the system-development life cycle, which contains multiple stages of development. This top-down, systematic approach to database development transforms business information requirements into an operational database.

Strategy and Analysis Phase

- Study and analyze the business requirements. Interview users and managers to identify the information requirements. Incorporate the enterprise and application mission statements as well as any future system specifications.
- Build models of the system. Transfer the business narrative into a graphical representation of business information needs and rules. Confirm and refine the model with the analysts and experts.

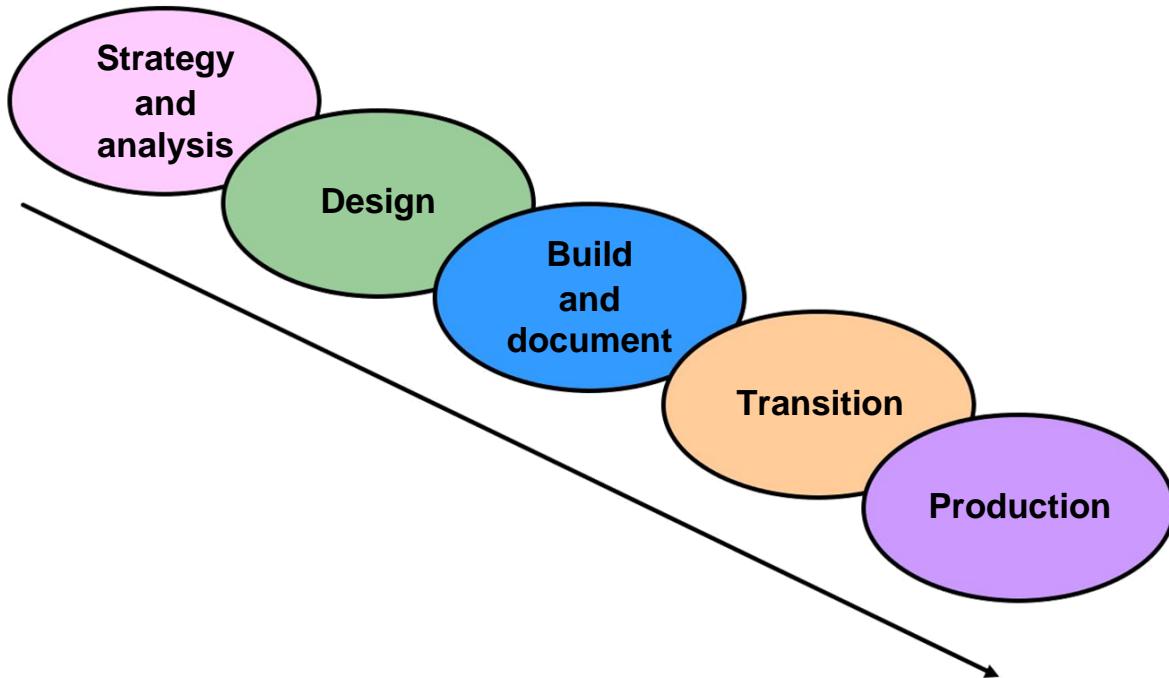
Design Phase

Design the database based on the model developed in the strategy and analysis phase.

Build and Documentation Phase

- Build the prototype system. Write and execute the commands to create the tables and supporting objects for the database.
- Develop user documentation, help text, and operations manuals to support the use and operation of the system.

System Development Life Cycle



ORACLE

Copyright © 2009, Oracle. All rights reserved.

System Development Life Cycle (continued)

Transition Phase

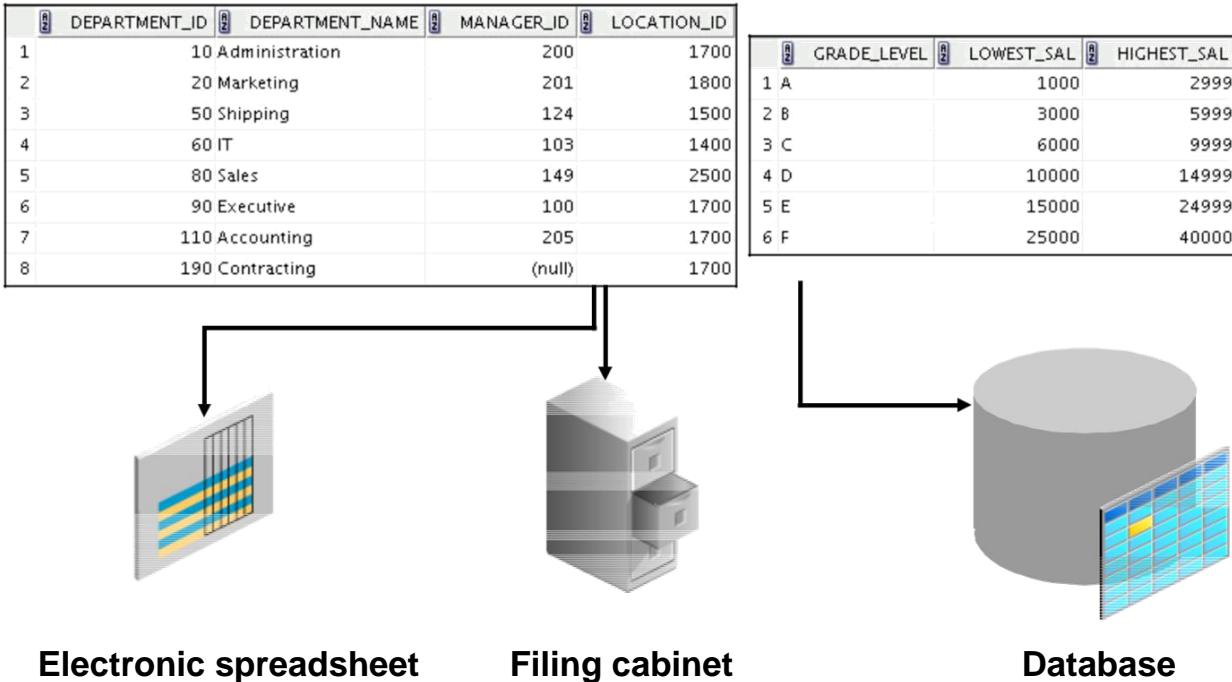
Refine the prototype. Move an application into production with user-acceptance testing, conversion of existing data, and parallel operations. Make any modifications required.

Production Phase

Roll out the system to the users. Operate the production system. Monitor its performance, and enhance and refine the system.

Note: The various phases of the system development life cycle can be carried out iteratively. This course focuses on the Build phase of the system development life cycle.

Data Storage on Different Media



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Storing Information

Every organization has some information needs. A library keeps a list of members, books, due dates, and fines. A company needs to save information about employees, departments, and salaries. These pieces of information are called *data*.

Organizations can store data on various media and in different formats, such as a hard-copy document in a filing cabinet or data stored in electronic spreadsheets or in databases.

A *database* is an organized collection of information.

To manage databases, you need a database management system (DBMS). A DBMS is a program that stores, retrieves, and modifies data in databases on request. There are four main types of databases: *hierarchical*, *network*, *relational*, and (most recently) *object relational*.

Relational Database Concept

- Dr. E. F. Codd proposed the relational model for database systems in 1970.
- It is the basis for the relational database management system (RDBMS).
- The relational model consists of the following:
 - Collection of objects or relations
 - Set of operators to act on the relations
 - Data integrity for accuracy and consistency



Copyright © 2009, Oracle. All rights reserved.

Relational Model

The principles of the relational model were first outlined by Dr. E. F. Codd in a June 1970 paper titled “A Relational Model of Data for Large Shared Data Banks.” In this paper, Dr. Codd proposed the relational model for database systems.

The common models used at that time were hierarchical and network, or even simple flat-file data structures. Relational database management systems (RDBMS) soon became very popular, especially for their ease of use and flexibility in structure. In addition, a number of innovative vendors, such as Oracle, supplemented the RDBMS with a suite of powerful application development and user products, providing a total solution.

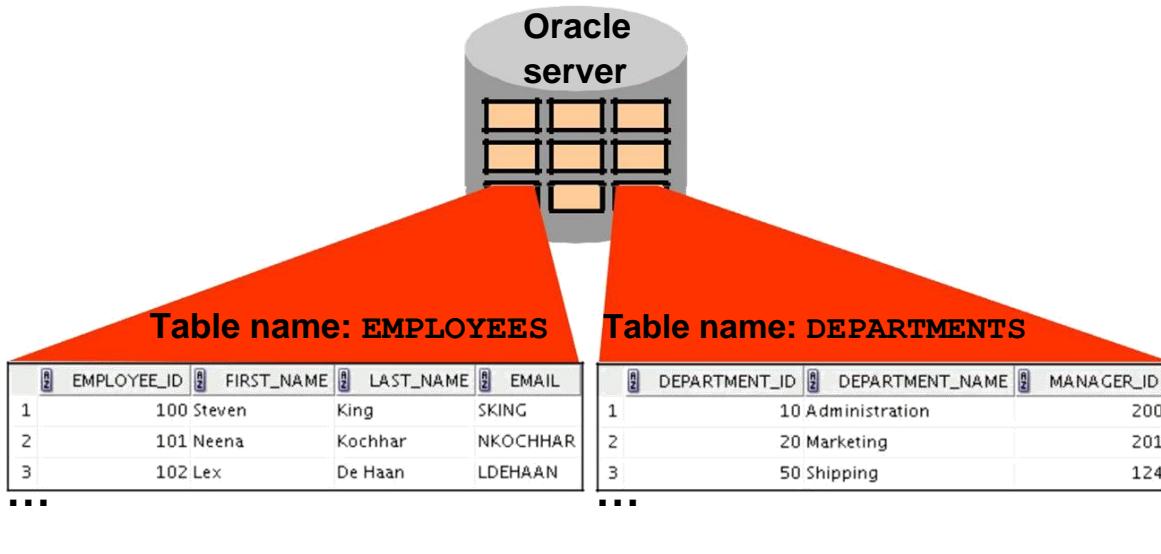
Components of the Relational Model

- Collections of objects or relations that store the data
- A set of operators that can act on the relations to produce other relations
- Data integrity for accuracy and consistency

For more information, see *An Introduction to Database Systems, Eighth Edition* (Addison-Wesley: 2004), written by Chris Date.

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.



ORACLE®

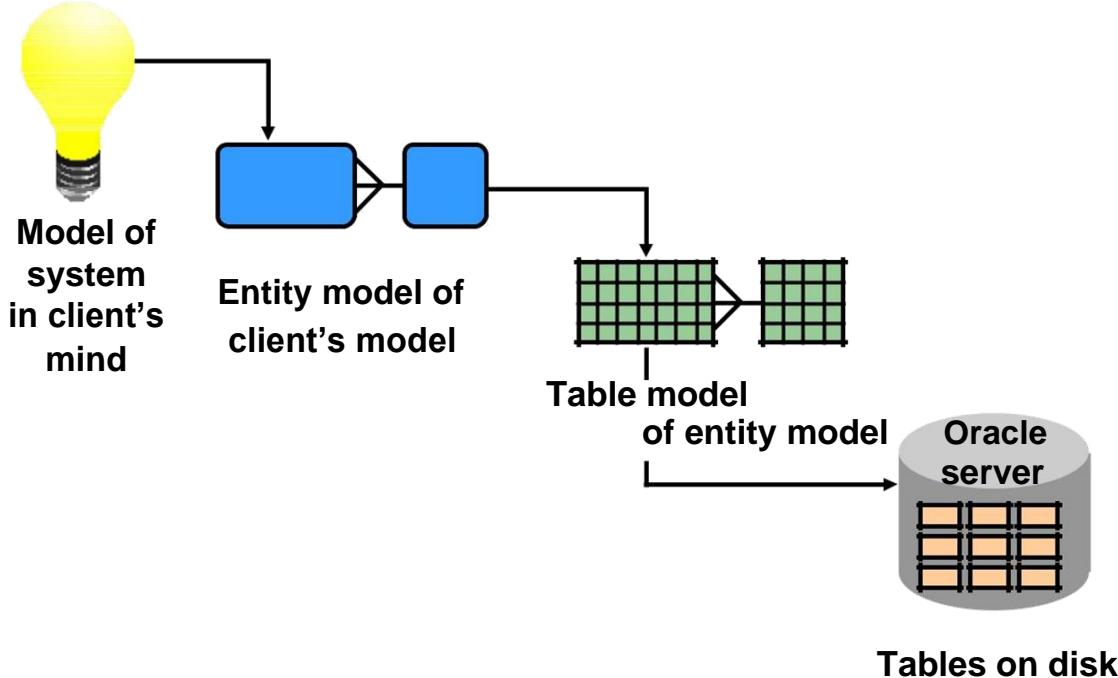
Copyright © 2009, Oracle. All rights reserved.

Definition of a Relational Database

A relational database uses relations or two-dimensional tables to store information.

For example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table, and a salary table.

Data Models



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Data Models

Models are a cornerstone of design. Engineers build a model of a car to work out any details before putting it into production. In the same manner, system designers develop models to explore ideas and improve the understanding of database design.

Purpose of Models

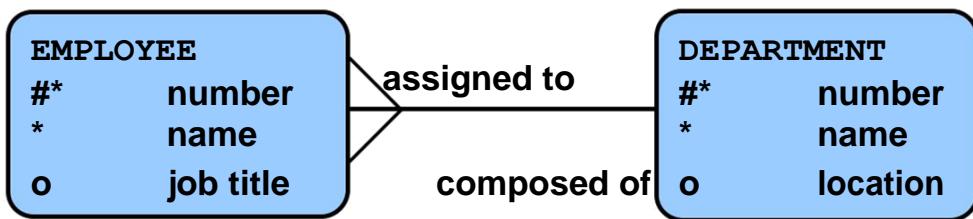
Models help communicate the concepts that are in people's minds. They can be used to do the following:

- Communicate
- Categorize
- Describe
- Specify
- Investigate
- Evolve
- Analyze
- Imitate

The objective is to produce a model that fits a multitude of these uses, can be understood by an end user, and contains sufficient detail for a developer to build a database system.

Entity Relationship Model

- Create an entity relationship diagram from business specifications or narratives:



- Scenario
 - “. . . Assign one or more employees to a department . . .”
 - “. . . Some departments do not yet have assigned employees . . .”

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ER Modeling

In an effective system, data is divided into discrete categories or entities. An entity relationship (ER) model is an illustration of various entities in a business and the relationships among them. An ER model is derived from business specifications or narratives and built during the analysis phase of the system development life cycle. ER models separate the information required by a business from the activities performed within a business. Although businesses can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant.

Benefits of ER Modeling

- Documents information for the organization in a clear, precise format
- Provides a clear picture of the scope of the information requirement
- Provides an easily understood pictorial map for database design
- Offers an effective framework for integrating multiple applications

ER Modeling (continued)

Key Components

- **Entity:** A thing of significance about which information needs to be known. Examples are departments, employees, and orders.
- **Attribute:** Something that describes or qualifies an entity. For example, for the employee entity, the attributes would be the employee number, name, job title, hire date, department number, and so on. Each of the attributes is either required or optional. This state is called *optionality*.
- **Relationship:** A named association between entities showing optionality and degree. Examples are employees and departments, and orders and items.

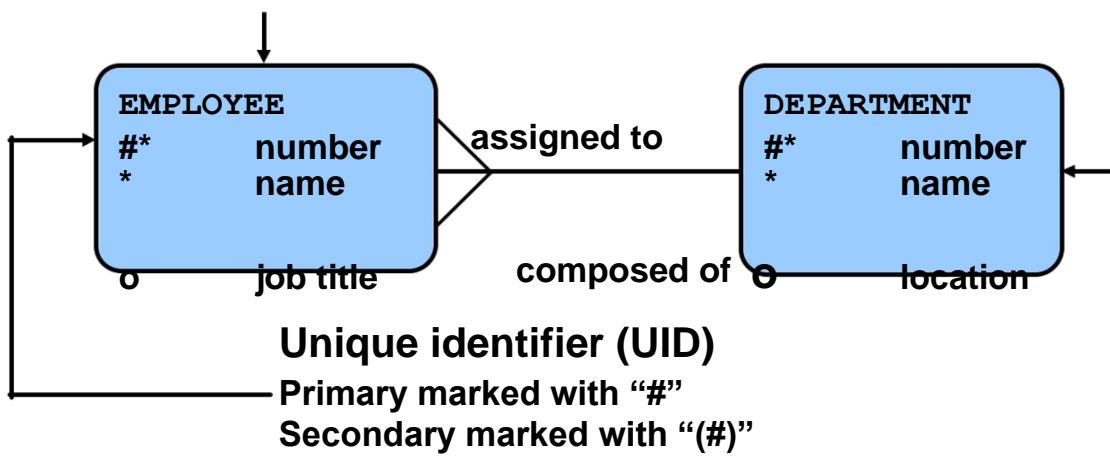
Entity Relationship Modeling Conventions

Entity

- Singular, unique name
- Uppercase
- Soft box
- Synonym in parentheses

Attribute

- Singular name
- Lowercase
- Mandatory marked with *
- Optional marked with “o”



ORACLE

Copyright © 2009, Oracle. All rights reserved.

ER Modeling Conventions

Entities

To represent an entity in a model, use the following conventions:

- Singular, unique entity name
- Entity name in uppercase
- Soft box
- Optional synonym names in uppercase within parentheses: ()

Attributes

To represent an attribute in a model, use the following conventions:

- Singular name in lowercase
- Asterisk (*) tag for mandatory attributes (that is, values that *must* be known)
- Letter “o” tag for optional attributes (that is, values that *may* be known)

Relationships

Symbol	Description
Dashed line	Optional element indicating “maybe”
Solid line	Mandatory element indicating “must be”
Crow’s foot	Degree element indicating “one or more”
Single line	Degree element indicating “one and only one”

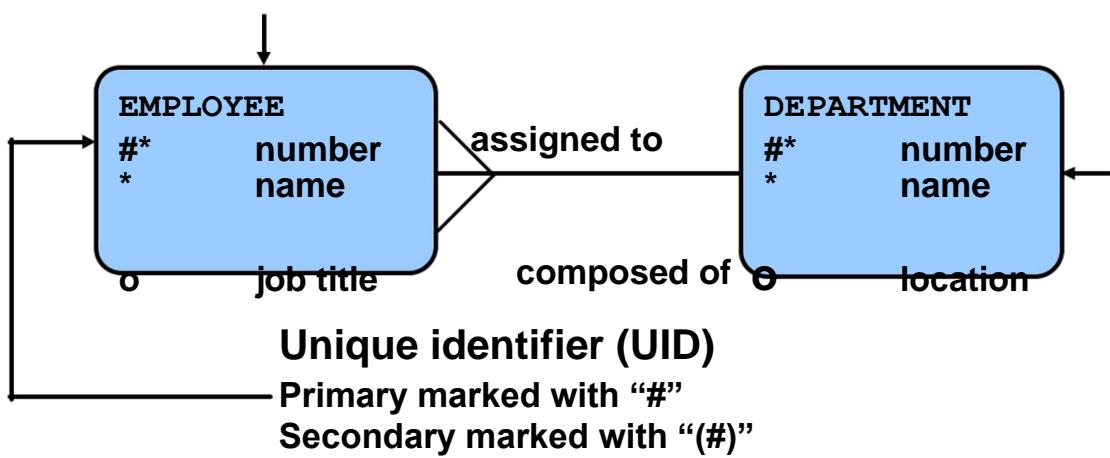
Entity Relationship Modeling Conventions

Entity

- Singular, unique name
- Uppercase
- Soft box
- Synonym in parentheses

Attribute

- Singular name
- Lowercase
- Mandatory marked with *
- Optional marked with “o”



ORACLE

Copyright © 2009, Oracle. All rights reserved.

ER Modeling Conventions (continued)

Relationships

Each direction of the relationship contains:

- **A label:** for example, *taught by* or *assigned to*
- **An optionality:** either *must be* or *maybe*
- **A degree:** either *one and only one* or *one or more*

Note: The term *cardinality* is a synonym for the term *degree*.

Each source entity {may be | must be} relationship name {one and only one | one or more} destination entity.

Note: The convention is to read clockwise.

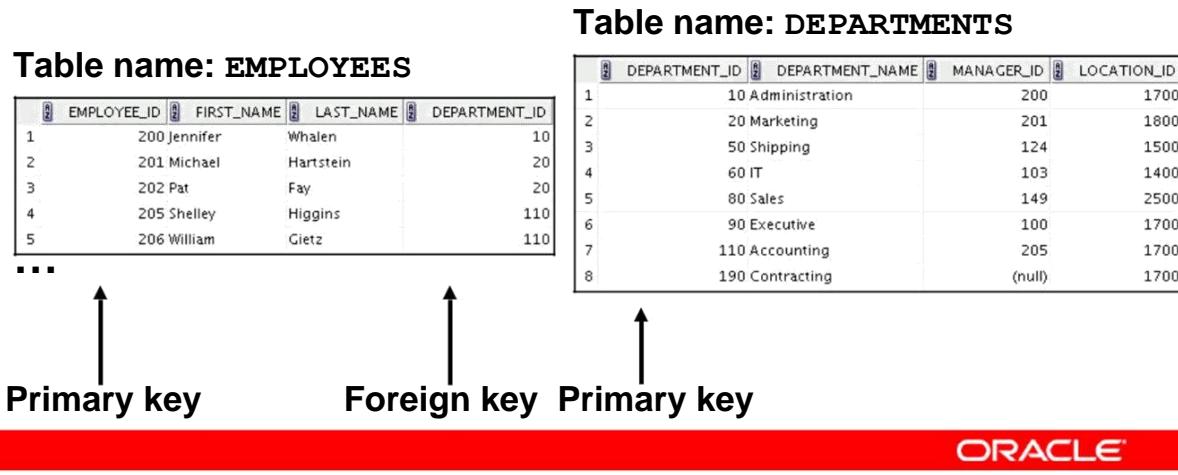
Unique Identifiers

A unique identifier (UID) is any combination of attributes or relationships, or both, that serves to distinguish occurrences of an entity. Each entity occurrence must be uniquely identifiable.

- **Tag each attribute that is part of the UID with a number sign: #**
- **Tag secondary UIDs with a number sign in parentheses: (#)**

Relating Multiple Tables

- Each row of data in a table is uniquely identified by a primary key (PK).
- You can logically relate data from multiple tables using foreign keys (FK).



Copyright © 2009, Oracle. All rights reserved.

ORACLE

Relating Multiple Tables

Each table contains data that describes exactly one entity. For example, the EMPLOYEES table contains information about employees. Categories of data are listed across the top of each table, and individual cases are listed below. Using a table format, you can readily visualize, understand, and use information.

Because data about different entities is stored in different tables, you may need to combine two or more tables to answer a particular question. For example, you may want to know the location of the department where an employee works. In this scenario, you need information from the EMPLOYEES table (which contains data about employees) and the DEPARTMENTS table (which contains information about departments). With an RDBMS, you can relate the data in one table to the data in another by using the foreign keys. A foreign key is a column (or a set of columns) that refers to a primary key in the same table or another table.

You can use the ability to relate data in one table to data in another to organize information in separate, manageable units. Employee data can be kept logically distinct from department data by storing it in a separate table.

Relating Multiple Tables (continued)

Guidelines for Primary Keys and Foreign Keys

- You cannot use duplicate values in a primary key.
- Primary keys generally cannot be changed.
- Foreign keys are based on data values and are purely logical (not physical) pointers.
- A foreign key value must match an existing primary key value or unique key value, or else it must be null.
- A foreign key must reference either a primary key or a unique key column.

Relational Database Terminology

The diagram shows the contents of the EMPLOYEES table from the Oracle database. The table has columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, COMMISSION_PCT, and DEPARTMENT_ID. Red boxes and green circles with numbers 1 through 6 highlight specific elements:

- 1**: A red box around the first column (EMPLOYEE_ID).
- 2**: A red box around the second column (FIRST_NAME).
- 3**: A red box around the third column (LAST_NAME).
- 4**: A red box around the fifth column (COMMISSION_PCT).
- 5**: A red box around the sixth column (DEPARTMENT_ID).
- 6**: A green circle with the number 6, centered over the row for employee 149 Eleni Zlotkey.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
1	200	Jennifer	Whalen	4400	(null)	10
2	201	Michael	Hartstein	13000	(null)	20
3	202	Pat	Fay	6000	(null)	20
4	205	Shelley	Higgins	12000	(null)	110
5	206	William	Gietz	8300	(null)	110
6	100	Steven	King	24000	(null)	90
7	101	Neena	Kochhar	17000	(null)	90
8	102	Lex	De Haan	17000	(null)	90
9	103	Alexander	Hunold	9000	(null)	60
10	104	Bruce	Ernst	6000	(null)	60
11	107	Diana	Lorentz	4200	(null)	60
12	124	Kevin	Mourgos	5800	(null)	50
13	141	Trenna	Rajs	3500	(null)	50
14	142	Curtis	Davies	3100	(null)	50
15	143	Randall	Matos	2600	(null)	50
16	144	Peter	Vargas	2500	(null)	50
17	149	Eleni	Zlotkey	10500	0.2	80
18	174	Ellen	Abel	11000	0.3	80
19	176	Jonathon	Taylor	8600	0.2	80
20	178	Kimberely	Grant	7000	0.15	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Terminology Used in a Relational Database

A relational database can contain one or many tables. A *table* is the basic storage structure of an RDBMS. A table holds all the data necessary about something in the real world, such as employees, invoices, or customers.

The slide shows the contents of the EMPLOYEES *table* or *relation*. The numbers indicate the following:

1. A single *row* (or *tuple*) representing all data required for a particular employee. Each row in a table should be identified by a primary key, which permits no duplicate rows. The order of rows is insignificant; specify the row order when the data is retrieved.
2. A *column* or attribute containing the employee number. The employee number identifies a *unique* employee in the EMPLOYEES table. In this example, the employee number column is designated as the *primary key*. A primary key must contain a value, and the value must be unique.
3. A column that is not a key value. A column represents one kind of data in a table; in this example, the data is the salaries of all the employees. Column order is insignificant when storing data; specify the column order when the data is retrieved.

Terminology Used in a Relational Database (continued)

4. A column containing the department number, which is also a *foreign key*. A foreign key is a column that defines how tables relate to each other. A foreign key refers to a primary key or a unique key in the same table or in another table. In the example, DEPARTMENT_ID *uniquely* identifies a department in the DEPARTMENTS table.
5. A *field* can be found at the intersection of a row and a column. There can be only one value in it.
6. A field may have no value in it. This is called a null value. In the EMPLOYEES table, only those employees who have the role of sales representative have a value in the COMMISSION_PCT (commission) field.

Relational Database Properties

A relational database:

- Can be accessed and modified by executing structured query language (SQL) statements
- Contains a collection of tables with no physical pointers
- Uses a set of operators



Copyright © 2009, Oracle. All rights reserved.

Properties of a Relational Database

In a relational database, you do not specify the access route to the tables, and you do not need to know how the data is arranged physically.

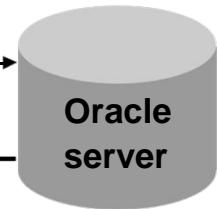
To access the database, you execute a structured query language (SQL) statement, which is the American National Standards Institute (ANSI) standard language for operating relational databases. The language contains a large set of operators for partitioning and combining relations. The database can be modified by using the SQL statements.

Communicating with an RDBMS Using SQL

SQL statement is entered.

```
SELECT department_name  
FROM   departments;
```

**Statement is sent to
Oracle server.**



ORACLE®

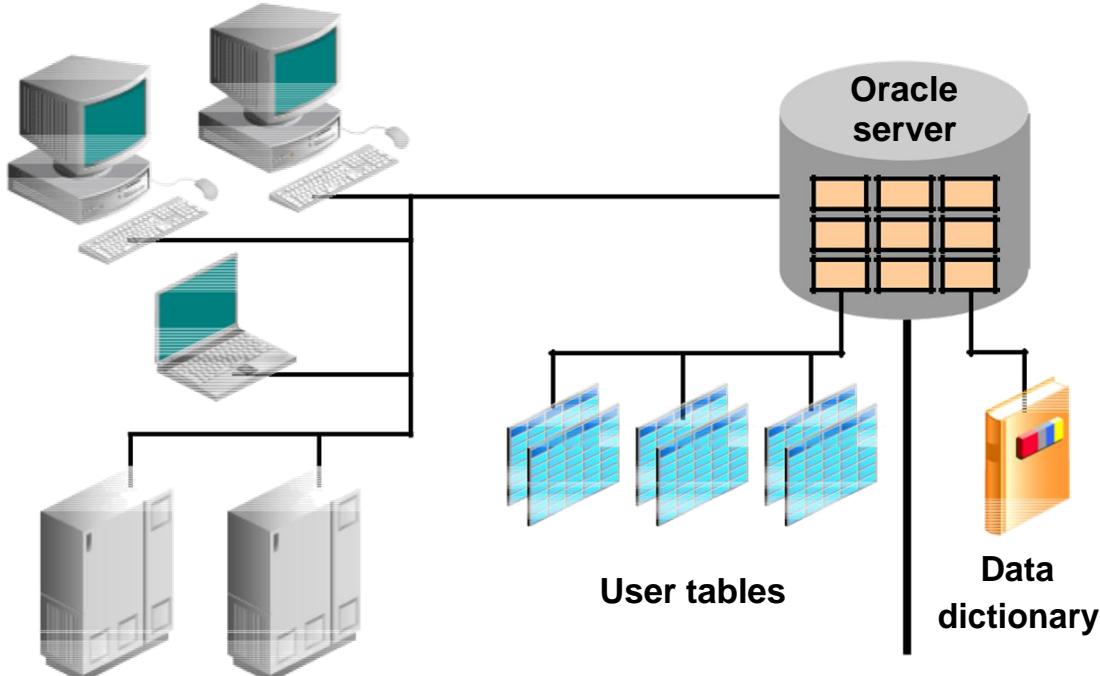
Copyright © 2009, Oracle. All rights reserved.

Structured Query Language

Using SQL, you can communicate with the Oracle server. SQL has the following advantages:

- Efficient
- Easy to learn and use
- Functionally complete (With SQL, you can define, retrieve, and manipulate data in the tables.)

Oracle's Relational Database Management System



ORACLE

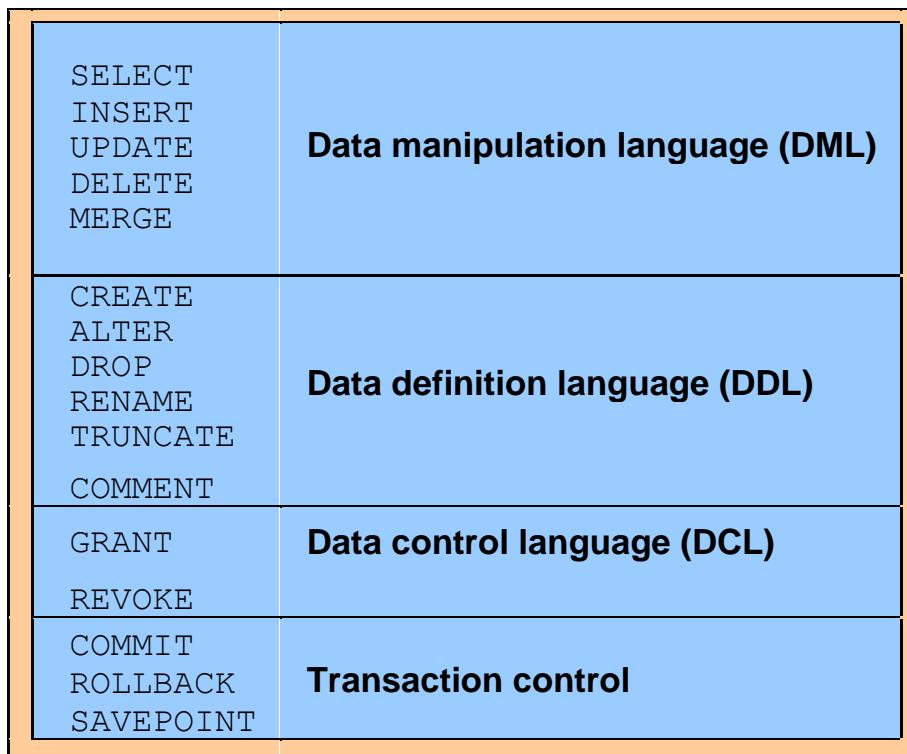
Copyright © 2009, Oracle. All rights reserved.

Oracle's Relational Database Management System

Oracle provides a flexible RDBMS called Oracle Database 10g. Using its features, you can store and manage data with all the advantages of a relational structure plus PL/SQL, an engine that provides you with the ability to store and execute program units. Oracle Database 10g also supports Java and XML. The Oracle server offers the options of retrieving data based on optimization techniques. It includes security features that control how a database is accessed and used. Other features include consistency and protection of data through locking mechanisms.

The Oracle10g release provides an open, comprehensive, and integrated approach to information management. An Oracle server consists of an Oracle Database and an Oracle server instance. Every time a database is started, a system global area (SGA) is allocated and Oracle background processes are started. The SGA is an area of memory that is used for database information shared by the database users. The combination of the background processes and memory buffers is called an Oracle *instance*.

SQL Statements



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Statements

Oracle SQL complies with industry-accepted standards. Oracle Corporation ensures future compliance with evolving standards by actively involving key personnel in SQL standards committees. Industry-accepted committees are American National Standards Institute (ANSI) and International Standards Organization (ISO). Both ANSI and ISO have accepted SQL as the standard language for relational databases.

Statement	Description
SELECT INSERT UPDATE DELETE MERGE	Retrieves data from the database, enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. Collectively known as <i>data manipulation language (DML)</i> .
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Sets up, changes, and removes data structures from tables. Collectively known as <i>data definition language (DDL)</i> .
GRANT REVOKE	Gives or removes access rights to both the Oracle database and the structures within it.
COMMIT ROLLBACK SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.

Tables Used in the Course

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400
2	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000
3	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
4	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
5	206	William	Gietz	WGIEWITZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300
6	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
7	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
8	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
9	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
10	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000
11	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200
12	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800
13	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500
14	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100
15	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600
16	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID	1344.429018	29-JAN-00	SA_MAN	10500
1	10	Administration	200	1700	1644.429267	11-MAY-96	SA REP	11000
2	20	Marketing	201	1800	1644.429265			
3	50	Shipping	124	1500	1644.429263			
4	60	IT	103	1400				
5	80	Sales	149	2500				
6	90	Executive	100	1700				
7	110	Accounting	205	1700				
8	190	Contracting	(null)	1700				

DEPARTMENTS

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

JOB_GRADES

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Tables Used in the Course

The following main tables are used in this course:

- EMPLOYEES table: Gives details of all the employees
- DEPARTMENTS table: Gives details of all the departments
- JOB_GRADES table: Gives details of salaries for various grades

Note: The structure and data for all the tables are provided in Appendix B.

Summary

- Oracle Database 10g is the database for grid computing.
- The database is based on the object relational database management system.
- Relational databases are composed of relations, managed by relational operations, and governed by data integrity constraints.
- With the Oracle server, you can store and manage information by using the SQL language and PL/SQL engine.



Copyright © 2009, Oracle. All rights reserved.

Summary

Relational database management systems are composed of objects or relations. They are managed by operations and governed by data integrity constraints.

Oracle Corporation produces products and services to meet your RDBMS needs. The main products are the following:

- Oracle Database 10g, with which you store and manage information by using SQL
- Oracle Application Server 10g, with which you run all of your applications
- Oracle Enterprise Manager 10g Grid Control, which you use to manage and automate administrative tasks across sets of systems in a grid environment

SQL

The Oracle server supports ANSI-standard SQL and contains extensions. SQL is the language that is used to communicate with the server to access, manipulate, and control data.

1

Retrieving Data Using the SQL SELECT Statement

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- List the capabilities of SQL SELECT statements
- Execute a basic SELECT statement
- Identify and use the key features of Oracle SQL Developer



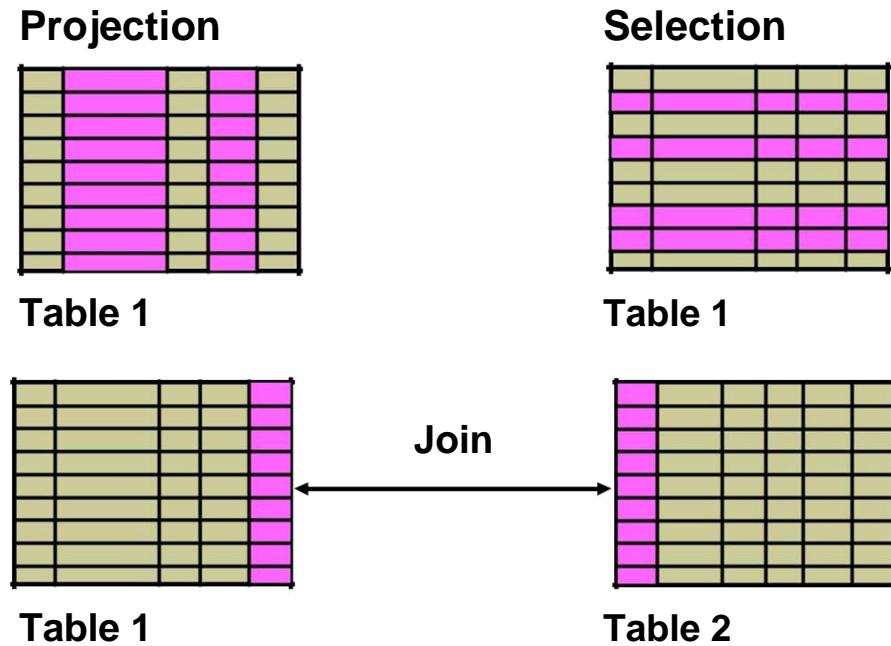
Copyright © 2009, Oracle. All rights reserved.

Objectives

To extract data from the database, you need to use the structured query language (SQL) SELECT statement. You may need to restrict the columns that are displayed. This lesson describes all the SQL statements that are needed to perform these actions. You may want to create SELECT statements that can be used more than once.

This lesson also covers the SQL Developer environment in which you execute SQL statements.

Capabilities of SQL SELECT Statements



Copyright © 2009, Oracle. All rights reserved.

ORACLE®

Capabilities of SQL SELECT Statements

A `SELECT` statement retrieves information from the database. With a `SELECT` statement, you can use the following capabilities:

- **Projection:** Choose the columns in a table that are returned by a query. Choose as few or as many of the columns as needed.
 - **Selection:** Choose the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.
 - **Joining:** Bring together data that is stored in different tables by specifying the link between them. SQL joins are covered in more detail in the lesson titled “Displaying Data from Multiple Tables.”

Basic SELECT Statement

```
SELECT * | { [DISTINCT] column|expression [alias], ... }
FROM table;
```

- `SELECT` identifies the columns to be displayed.
- `FROM` identifies the table containing those columns.



Copyright © 2009, Oracle. All rights reserved.

Basic SELECT Statement

In its simplest form, a `SELECT` statement must include the following:

- A `SELECT` clause, which specifies the columns to be displayed
- A `FROM` clause, which identifies the table containing the columns that are listed in the `SELECT` clause

In the syntax:

<code>SELECT</code>	is a list of one or more columns
<code>*</code>	selects all columns
<code>DISTINCT</code>	suppresses duplicates
<code>column expression</code>	selects the named column or the expression
<code>alias</code>	gives selected columns different headings
<code>FROM table</code>	specifies the table containing the columns

Note: Throughout this course, the words *keyword*, *clause*, and *statement* are used as follows:

- A *keyword* refers to an individual SQL element.
For example, `SELECT` and `FROM` are keywords.
- A *clause* is a part of a SQL statement.
For example, `SELECT employee_id, last_name, ...` is a clause.
- A *statement* is a combination of two or more clauses.
For example, `SELECT * FROM employees` is a SQL statement.

Selecting All Columns

```
SELECT *  
FROM departments;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700



Copyright © 2009, Oracle. All rights reserved.

Selecting All Columns of All Rows

You can display all columns of data in a table by following the SELECT keyword with an asterisk (*). In the example in the slide, the department table contains four columns: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The table contains eight rows, one for each department.

You can also display all columns in the table by listing all the columns after the SELECT keyword. For example, the following SQL statement (like the example in the slide) displays all columns and all rows of the DEPARTMENTS table:

```
SELECT department_id, department_name, manager_id, location_id  
FROM departments;
```

Selecting Specific Columns

```
SELECT department_id, location_id  
FROM   departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700



Copyright © 2009, Oracle. All rights reserved.

Selecting Specific Columns of All Rows

You can use the SELECT statement to display specific columns of the table by specifying the column names, separated by commas. The example in the slide displays all the department numbers and location numbers from the DEPARTMENTS table.

In the SELECT clause, specify the columns that you want, in the order in which you want them to appear in the output. For example, to display location before department number going from left to right, you use the following statement:

```
SELECT location_id, department_id  
FROM   departments;
```

	LOCATION_ID	DEPARTMENT_ID
1	1700	10
2	1800	20
3	1500	50
4	1400	60

...

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.
- In SQL Developer, SQL statements can optionally be terminated by a semicolon (;). Semicolons are required if you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).



Copyright © 2009, Oracle. All rights reserved.

Writing SQL Statements

Using the following simple rules and guidelines, you can construct valid statements that are both easy to read and easy to edit:

- SQL statements are not case-sensitive (unless indicated).
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.
- Keywords typically are entered in uppercase; all other words, such as table names and columns, are entered in lowercase.

Executing SQL Statements

Using SQL Developer, click the Execute button to run the command or commands in the editing window.

Using SQL*Plus, terminate the SQL statement with a semicolon and then press the Enter key to run the command.

Column Heading Defaults

- SQL Developer:
 - Default heading alignment: Center
 - Default heading display: Uppercase
- SQL*Plus:
 - Character and Date column headings are left-aligned
 - Number column headings are right-aligned
 - Default heading display: Uppercase

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

Column Heading Defaults

In SQL Developer, column headings are displayed in uppercase and centered.

```
SELECT last_name, hire_date, salary  
FROM employees;
```

LAST_NAME	HIRE_DATE	SALARY
Whalen	17-SEP-87	4400
Hartstein	17-FEB-96	13000
Fay	17-AUG-97	6000
Higgins	07-JUN-94	12000
Gietz	07-JUN-94	8300
King	17-JUN-87	24000
Kochhar	21-SEP-89	17000
De Haan	13-JAN-93	17000
...		

You can override the column heading display with an alias. Column aliases are covered later in this lesson.

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide



Copyright © 2009, Oracle. All rights reserved.

Arithmetic Expressions

You may need to modify the way in which data is displayed, or you may want to perform calculations or look at what-if scenarios. These are all possible using arithmetic expressions. An arithmetic expression can contain column names, constant numeric values, and the arithmetic operators.

Arithmetic Operators

The slide lists the arithmetic operators that are available in SQL. You can use arithmetic operators in any clause of a SQL statement (except the FROM clause).

Note: With the DATE and TIMESTAMP data types, you can use the addition and subtraction operators only.

Using Arithmetic Operators

```
SELECT last_name, salary, salary +
300 FROM employees;
```

	LAST_NAME	SALARY	SALARY+300
1	Whalen	4400	4700
2	Hartstein	13000	13300
3	Fay	6000	6300
4	Higgins	12000	12300
5	Gietz	8300	8600
6	King	24000	24300
7	Kochhar	17000	17300
8	De Haan	17000	17300
9	Hunold	9000	9300
10	Ernst	6000	6300
...			



Copyright © 2009, Oracle. All rights reserved.

Using Arithmetic Operators

The example in the slide uses the addition operator to calculate a salary increase of \$300 for all employees. The slide also displays a SALARY+300 column in the output.

Note that the resultant calculated column SALARY+300 is not a new column in the EMPLOYEES table; it is for display only. By default, the name of a new column comes from the calculation that generated it—in this case, salary+300.

Note: The Oracle server ignores blank spaces before and after the arithmetic operator.

Operator Precedence

If an arithmetic expression contains more than one operator, multiplication and division are evaluated first. If operators in an expression are of the same priority, then evaluation is done from left to right. You can use parentheses to force the expression that is enclosed by parentheses to be evaluated first.

Rules of Precedence:

- Multiplication and division occur before addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to override the default precedence or to clarify the statement.

Operator Precedence

```
SELECT last_name, salary, 12*salary+100
FROM employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	Whalen	4400	52900
2	Hartstein	13000	156100
3	Fay	6000	72100

```
SELECT last_name, salary, 12*(salary+100)
FROM employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	Whalen	4400	54000
2	Hartstein	13000	157200
3	Fay	6000	73200

Copyright © 2009, Oracle. All rights reserved.

Operator Precedence (continued)

The first example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation by multiplying the monthly salary by 12, plus a one-time bonus of \$100. Note that multiplication is performed before addition.

Note: Use parentheses to reinforce the standard order of precedence and to improve clarity. For example, the expression in the slide can be written as $(12 * \text{salary}) + 100$ with no change in the result.

Using Parentheses

You can override the rules of precedence by using parentheses to specify the desired order in which operators are to be executed.

The second example in the slide displays the last name, salary, and annual compensation of employees. It calculates the annual compensation as follows: adding a monthly bonus of \$100 to the monthly salary, and then multiplying that subtotal by 12. Because of the parentheses, addition takes priority over multiplication.

Defining a Null Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A null is not the same as a zero or a blank space.

```
SELECT last_name, job_id, salary,
commission_pct FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	Whalen	AD_ASST	4400	(null)
2	Hartstein	MK_MAN	13000	(null)
3	Fay	MK_REP	6000	(null)
...				
17	Zlotkey	SA_MAN	10500	0.2
18	Abel	SA_REP	11000	0.3
19	Taylor	SA_REP	8600	0.2
20	Grant	SA_REP	7000	0.15



Copyright © 2009, Oracle. All rights reserved.

Null Values

If a row lacks a data value for a particular column, that value is said to be *null* or to contain a null.

A null is a value that is unavailable, unassigned, unknown, or inapplicable. A null is not the same as a zero or a space. Zero is a number, and a space is a character.

Columns of any data type can contain nulls. However, some constraints (NOT NULL and PRIMARY KEY) prevent nulls from being used in the column.

In the COMMISSION_PCT column in the EMPLOYEES table, notice that only a sales manager or sales representative can earn a commission. Other employees are not entitled to earn commissions. A null represents that fact.

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

```
SELECT last_name,  
       12*salary*commission_pct FROM employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
Whalen	(null)
Hartstein	(null)
Fay	(null)
Higgins	(null)
...	
Zlotkey	25200
Abel	39600
Taylor	20640
Grant	12600

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Null Values in Arithmetic Expressions

If any column value in an arithmetic expression is null, the result is null. For example, if you attempt to perform division by zero, you get an error. However, if you divide a number by null, the result is a null or unknown.

In the example in the slide, employee King does not get any commission. Because the COMMISSION_PCT column in the arithmetic expression is null, the result is null.

For more information, see “Basic Elements of SQL” in *SQL Reference*.

Defining a Column Alias

A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name (There can also be the optional `AS` keyword between the column name and alias.)
- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive



Copyright © 2009, Oracle. All rights reserved.

Column Aliases

When displaying the result of a query, SQL Developer normally uses the name of the selected column as the column heading. This heading may not be descriptive and, therefore, maybe difficult to understand. You can change a column heading by using a column alias.

Specify the alias after the column in the `SELECT` list using a space as a separator. By default, alias headings appear in uppercase. If the alias contains spaces or special characters (such as # or \$), or if it is case-sensitive, enclose the alias in double quotation marks ("").

Using Column Aliases

```
SELECT last_name AS name, commission_pct comm
FROM employees;
```

	NAME	COMM
1	Whalen	(null)
2	Hartstein	(null)
3	Fay	(null)

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM employees;
```

	Name	Annual Salary
1	Whalen	52800
2	Hartstein	156000
3	Fay	72000

...

Copyright © 2009, Oracle. All rights reserved.

Column Aliases (continued)

The first example displays the names and the commission percentages of all the employees. Notice that the optional AS keyword has been used before the column alias name. The result of the query is the same whether the AS keyword is used or not. Also notice that the SQL statement has the column aliases, name and comm, in lowercase, whereas the result of the query displays the column headings in uppercase. As mentioned in a previous slide, column headings appear in uppercase by default.

The second example displays the last names and annual salaries of all the employees. Because Annual Salary contains a space, it has been enclosed in double quotation marks. Notice that the column heading in the output is exactly the same as the column alias.

Concatenation Operator

A concatenation operator:

- Links columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression

```
SELECT last_name || job_id AS "Employees"  
FROM employees;
```

Employees
1 AbelSA_REP
2 DaviesST_CLERK
3 De HaanAD_VP
4 ErnstIT_PROG
...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Concatenation Operator

You can link columns to other columns, arithmetic expressions, or constant values to create a character expression by using the *concatenation operator* (||). Columns on either side of the operator are combined to make a single output column.

In the example, LAST_NAME and JOB_ID are concatenated, and they are given the alias Employees. Notice that the employee last name and job code are combined to make a single output column.

The AS keyword before the alias name makes the SELECT clause easier to read.

Null Values with the Concatenation Operator

If you concatenate a null value with a character string, the result is a character string. LAST_NAME || NULL results in LAST_NAME.

Literal Character Strings

- A literal is a character, a number, or a date that is included in the SELECT statement.
- Date and character literal values must be enclosed by single quotation marks.
- Each character string is output once for each row returned.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Literal Character Strings

A literal is a character, a number, or a date that is included in the SELECT list and that is not a column name or a column alias. It is printed for each row returned. Literal strings of free-format text can be included in the query result and are treated the same as a column in the SELECT list.

Date and character literals *must* be enclosed by single quotation marks (' '); number literals need not be so enclosed.

Using Literal Character Strings

```
SELECT last_name || ' is a ' || job_id
      AS "Employee Details"
  FROM employees;
```

Employees Details
1 Abel is a SA_REP
2 Davies is a ST_CLERK
3 De Haan is a AD_VP
4 Ernst is a IT_PROG
5 Fay is a MK_REP
6 Gietz is a AC_ACCOUNT
7 Grant is a SA_REP
8 Hartstein is a MK_MAN
...



Copyright © 2009, Oracle. All rights reserved.

Literal Character Strings (continued)

The example in the slide displays last names and job codes of all employees. The column has the heading Employee Details. Notice the spaces between the single quotation marks in the SELECT statement. The spaces improve the readability of the output.

In the following example, the last name and salary for each employee are concatenated with a literal to give the returned rows more meaning:

```
SELECT last_name || ': 1 Month salary = ' || salary Monthly
      FROM employees;
```

MONTHLY
1 Whalen: 1 Month salary = 4400
2 Hartstein: 1 Month salary = 13000
3 Fay: 1 Month salary = 6000
4 Higgins: 1 Month salary = 12000
5 Gietz: 1 Month salary = 8300
6 King: 1 Month salary = 24000
7 Kochhar: 1 Month salary = 17000
...

Alternative Quote (q) Operator

- Specify your own quotation mark delimiter.
- Choose any delimiter.
- Increase readability and usability.

```
SELECT department_name ||  
      q'[ , it's assigned Manager Id: ]'  
      || manager_id  
      AS "Department and Manager"  
FROM departments;
```

Department and Manager
1 Administration, it's assigned Manager Id: 200
2 Marketing, it's assigned Manager Id: 201
3 Shipping, it's assigned Manager Id: 124

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Alternative Quote (q) Operator

Many SQL statements use character literals in expressions or conditions. If the literal itself contains a single quotation mark, you can use the quote (q) operator and choose your own quotation mark delimiter.

You can choose any convenient delimiter, single-byte or multibyte, or any of the following character pairs: [], { }, (), or <>.

In the example shown, the string contains a single quotation mark, which is normally interpreted as a delimiter of a character string. By using the q operator, however, the brackets [] are used as the quotation mark delimiter. The string between the brackets delimiters is interpreted as a literal character string.

Duplicate Rows

The default display of queries is all rows, including duplicate rows.

1

```
SELECT department_id  
FROM employees;
```

	DEPARTMENT_ID
1	10
2	20
3	20
4	110
5	110

...

2

```
SELECT DISTINCT department_id  
FROM employees;
```

	DEPARTMENT_ID
1	(null)
2	20
3	90
4	110
5	50

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Duplicate Rows

Unless you indicate otherwise, SQL Developer displays the results of a query without eliminating duplicate rows. The first example in the slide displays all the department numbers from the EMPLOYEES table. Notice that the department numbers are repeated.

To eliminate duplicate rows in the result, include the DISTINCT keyword in the SELECT clause immediately after the SELECT keyword. In the second example in the slide, the EMPLOYEES table actually contains 20 rows, but there are only seven unique department numbers in the table.

You can specify multiple columns after the DISTINCT qualifier. The DISTINCT qualifier affects all the selected columns, and the result is every distinct combination of the columns.

```
SELECT DISTINCT department_id, job_id  
FROM employees;
```

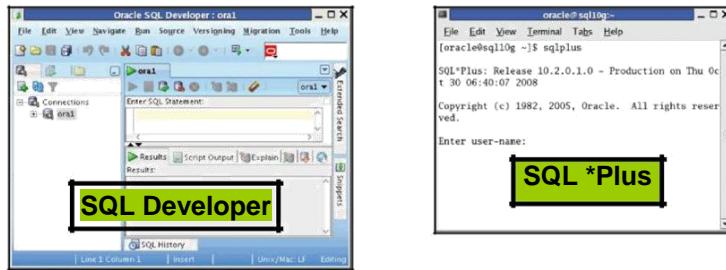
DEPARTMENT_ID	JOB_ID
1	110 AC_ACCOUNT
2	90 AD_VP
3	50 ST_CLERK

...

Development Environments for SQL

In this course:

- Primarily use Oracle SQL Developer 1.5.1
- Use SQL*Plus:
 - In case you do not have access to Oracle SQL Developer
 - Or when any command does not run in Oracle SQL Developer



ORACLE®

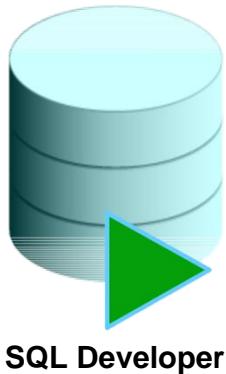
Copyright © 2009, Oracle. All rights reserved.

Development Environments for SQL

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. For commands that are not supported by Oracle SQL Developer, use the SQL*Plus environment.

What Is Oracle SQL Developer?

- Oracle SQL Developer is a graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle Database schema by using the standard Oracle Database authentication.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

What Is Oracle SQL Developer?

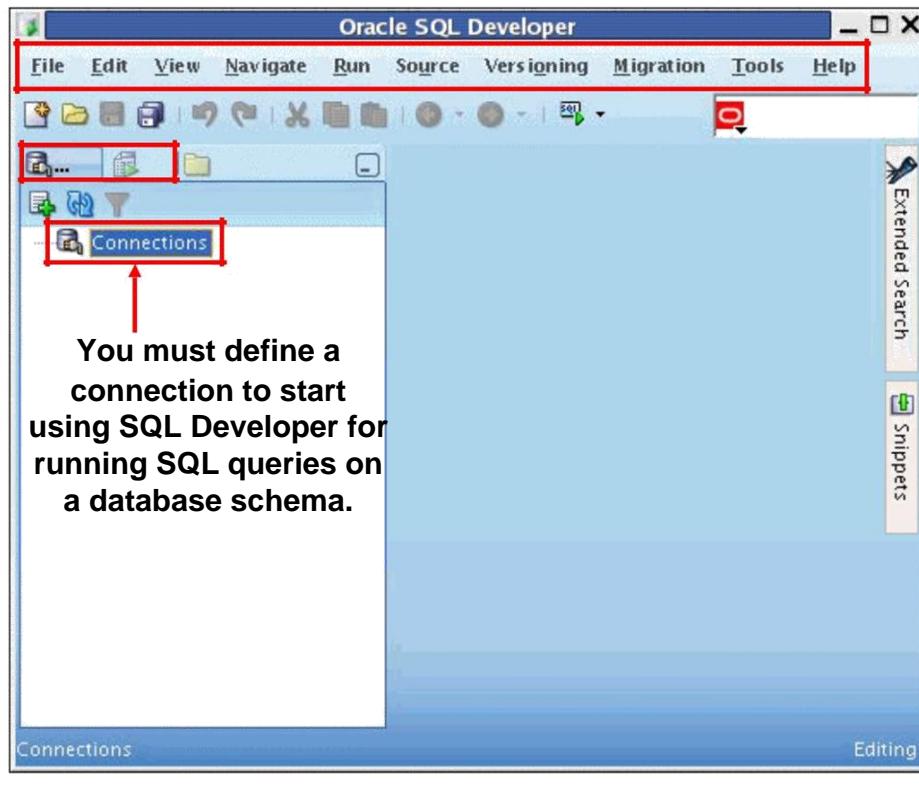
Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

Oracle 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 by using the standard Oracle Database authentication. When connected, you can perform operations on objects in the database.

Oracle SQL Developer Interface



Oracle SQL Developer Interface

Oracle SQL Developer has two main navigation tabs:

- **Connections tab:** By using this tab, you can browse database objects and users to which you have access.
- **Reports tab:** By using this tab, you can run predefined reports, or create and add your own reports.

Oracle SQL Developer uses the left pane for navigation to find and select objects, and the right pane to display information about selected objects. You can customize many aspects of the appearance and behavior of Oracle SQL Developer by setting preferences. The menus at the top contain standard entries, plus entries for features specific to Oracle SQL Developer.

Note: You must define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures/functions.

Creating a Database Connection

- You must have at least one database connection to use Oracle SQL Developer.
- You can create and test connections for:
 - Multiple databases
 - Multiple schemas
- Oracle SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an XML file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Database Connection

A connection is an Oracle SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use Oracle SQL Developer, you must have at least one database connection, which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

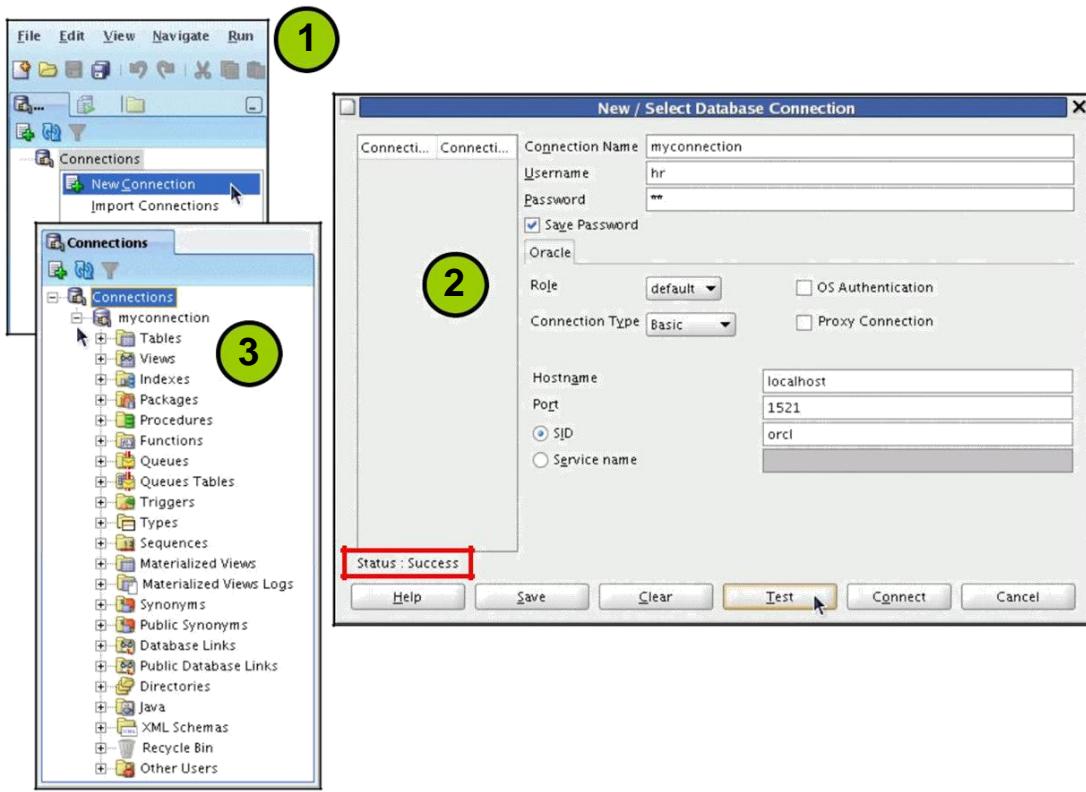
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory. But, it can also be in the directory specified by the `TNS_ADMIN` environment variable or the registry value. When you start Oracle SQL Developer and display the Database Connections dialog box, Oracle SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

Note: On Windows systems, if the `tnsnames.ora` file exists but Oracle SQL Developer is not using its connections, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it later.

You can create additional connections as different users to the same database or to connect to the different databases.

Creating a Database Connection



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Database Connection (continued)

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New/Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
 1. From the Role drop-down list, you can select either *default* or SYSDBA (you will select SYSDBA for the *sys* user or any user with DBA privileges).
 2. You can select the connection type as:
 - **Basic:** In this type, you enter the host name and system identifier (SID) for the database that you want to connect to. The Port is already set to 1521. Or, you can also enter the Service name directly if you are using a remote database connection.
 - **TNS:** You select any one of the database aliases imported from the *tnsnames.ora* file
 - **Advanced:** You define a custom JDBC URL to connect to the database.
3. Click Test to make sure that the connection has been set correctly.
4. Click Connect.

If you select the Save Password check box, the password is saved to an XML file. So, after you close the Oracle SQL Developer connection and open it again, you will not be prompted for the password.

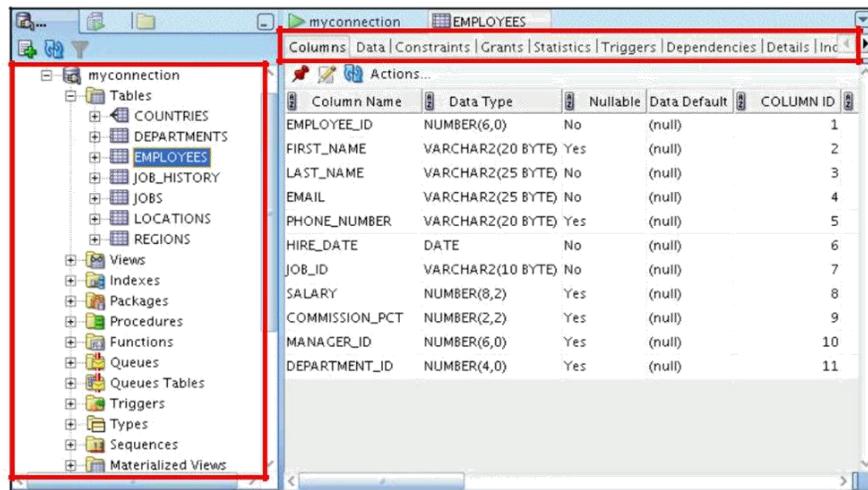
Creating a Database Connection (continued)

3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions, for example, dependencies, details, statistics, and so on.

Browsing Database Objects

Use the Connections Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Browsing Database Objects

After you have created a database connection, you can use the Connections Navigator to browse through many objects in a database schema including Tables, Views, Indexes, Packages, Procedures, Triggers, Types, and so on.

Oracle SQL Developer uses the left pane for navigation to find and select objects and the right pane to display information about the selected objects. You can customize many aspects of the appearance of Oracle SQL Developer by setting preferences.

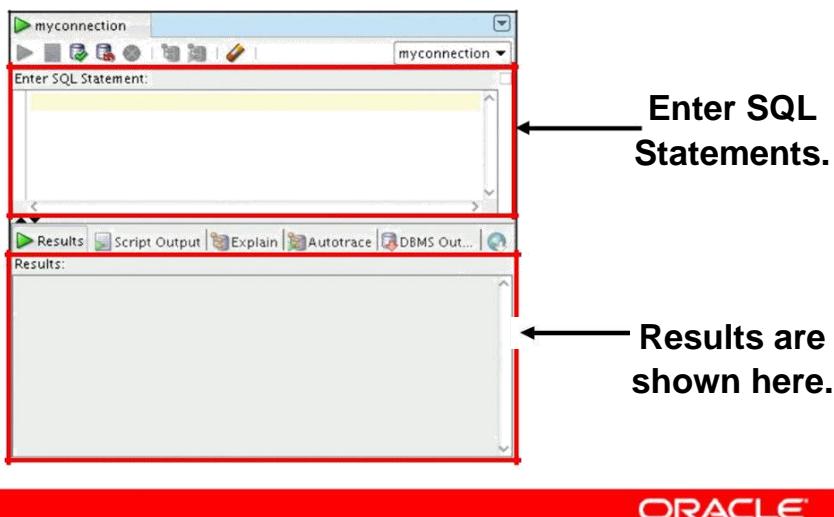
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, the details about columns, constraints, grants, statistics, triggers, and so on are displayed in an easy-to-read tabbed page.

If you want to see the definition of the EMPLOYEES table as shown in the slide, perform the following steps:

1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. By using the Data tab, you can view the data in the table and also enter new rows, update data, and commit these changes to the database.

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements.
- Specify any actions that can be processed by the database connection associated with the Worksheet.



Copyright © 2009, Oracle. All rights reserved.

Using the SQL Worksheet

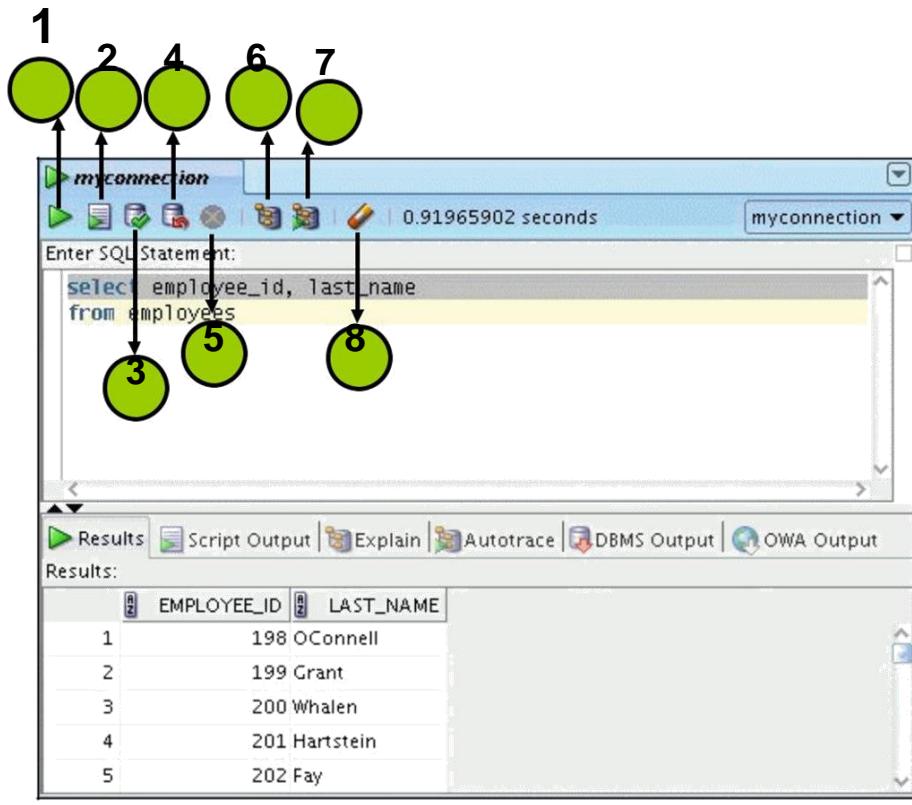
When you connect to a database, a SQL Worksheet window for that connection is automatically opened. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle Database. However, the SQL*Plus commands used in Oracle SQL Developer have to be interpreted by the SQL Worksheet before being passed to the database.

The SQL Worksheet currently supports a number of SQL*Plus commands. Those commands that are not supported by the SQL Worksheet are ignored and not sent to the Oracle Database. Through the SQL Worksheet, you can execute SQL statements and some of the SQL*Plus commands.

You can display a SQL Worksheet by using any of the following two options:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon available on the main toolbar.

Using the SQL Worksheet



Copyright © 2009, Oracle. All rights reserved.

Using the SQL Worksheet (continued)

You may want to use shortcut keys or icons to perform certain tasks, such as executing a SQL statement, running a script, or viewing the history of the SQL statements that you have executed.

You can use the SQL Worksheet toolbar that contains icons to perform the following tasks:

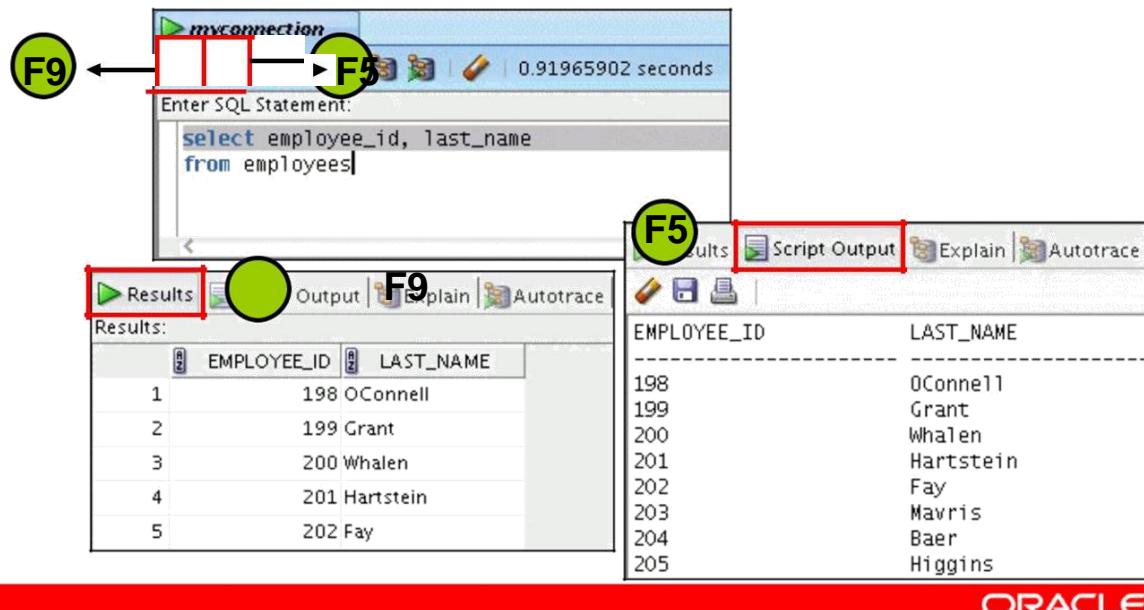
- Execute Statement:** This executes the statement at the cursor in the Enter SQL Statement box. Alternatively, you can press [F9]. The output is generally shown in a formatted manner in the Results tab page.
- Run Script:** This executes all statements in the Enter SQL Statement box using the Script Runner. The output is generally shown in the conventional script format in the Scripts tab page.
- Commit:** This writes any changes to the database and ends the transaction.
- Rollback:** This discards any changes to the database, without writing them to the database, and ends the transaction.
- Cancel:** This stops the execution of any statements currently being executed.
- Execute Explain Plan:** This generates the execution plan, which you can see by clicking the Explain tab.

Using the SQL Worksheet (continued)

7. **Autotrace:** This displays trace-related information when you execute the SQL statement by clicking the Autotrace icon. This information can help you to identify the SQL statements that will benefit from tuning.
8. **Clear:** This erases the statement or statements in the Enter SQL Statement box. Alternatively, press and hold [Ctrl] + [D] to erase the statements.

Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.



Copyright © 2009, Oracle. All rights reserved.

Executing SQL Statements

In the SQL Worksheet, you can use the Enter SQL Statement box to enter a single or multiple SQL statements. For a single statement, the semicolon at the end is optional.

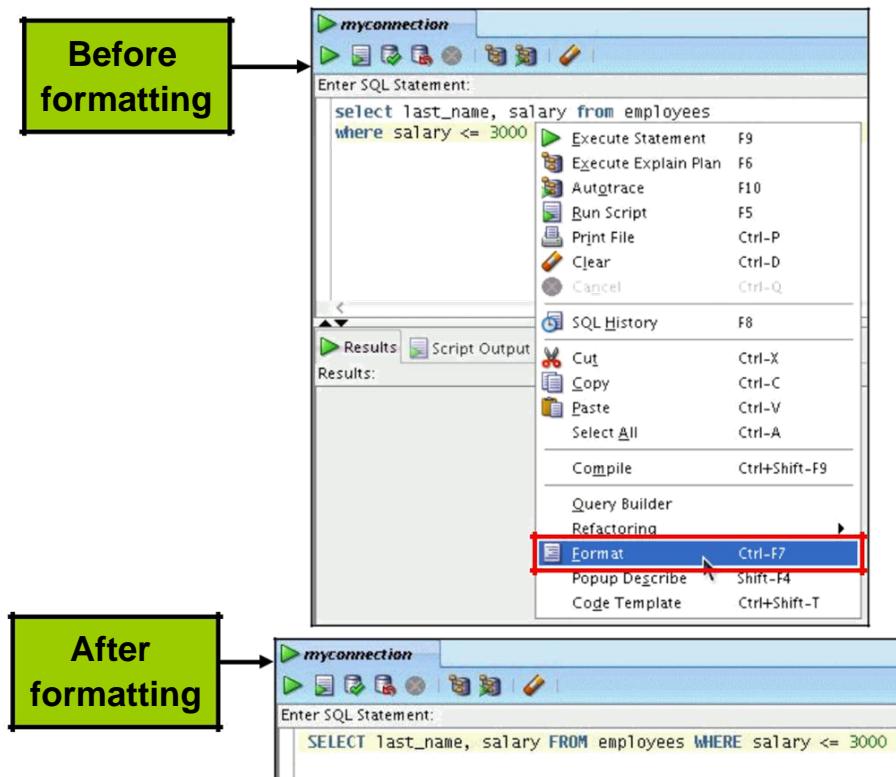
When you enter the statement, the SQL keywords are automatically highlighted. To execute a SQL statement, ensure that your cursor is within the statement and click the Execute Statement icon.

Alternatively, you can press [F9].

To execute multiple SQL statements and see the results, click the Run Script icon. Alternatively, you can press [F5].

The example in the slide shows the difference in output for the same query when F9 key or Execute Statement is used versus the output when F5 or Run Script is used.

Formatting the SQL Code



ORACLE

Copyright © 2009, Oracle. All rights reserved.

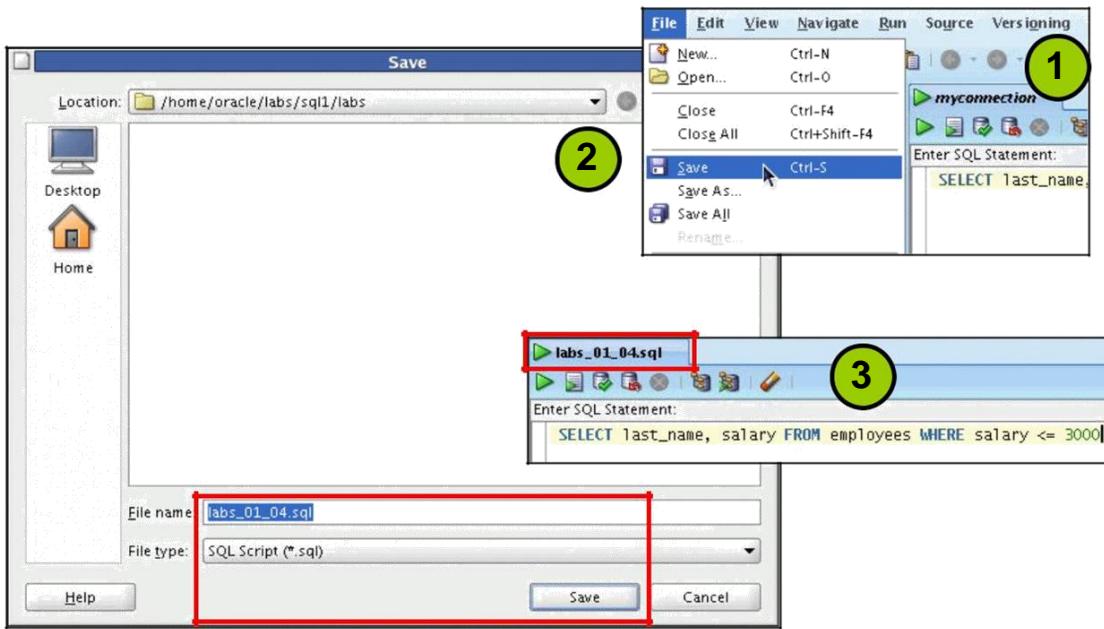
Formatting the SQL Code

You may want to beautify the indentation, spacing, capitalization, and the line separation of the SQL code. Oracle SQL Developer has the feature for formatting the SQL code.

To format the SQL code, right-click in the statement area, and select Format SQL.

In the example in the slide, before formatting, the keywords are not capitalized and the statement is not properly indented in the SQL code. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented, if needed.

Saving SQL Statements



ORACLE®

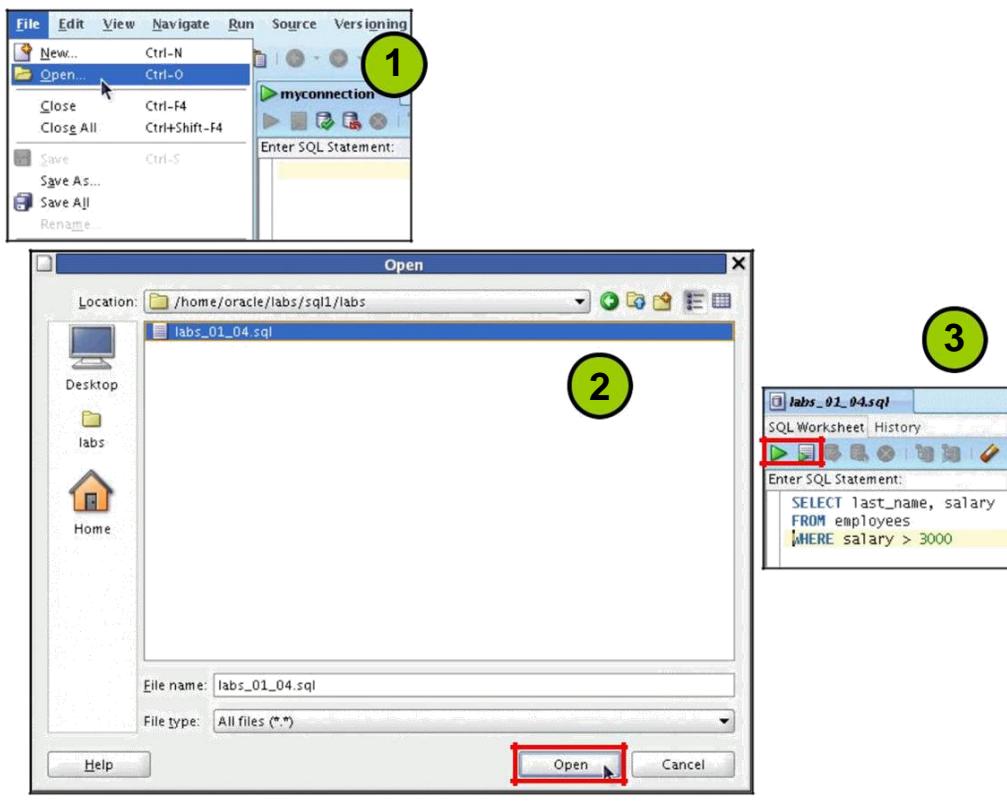
Copyright © 2009, Oracle. All rights reserved.

Saving SQL Statements

In most of the practices that you will perform, you will need to save a particular query in the SQL Worksheet as a .sql file. To do so, perform the following:

- From the File menu, select Save or Save As (if you are renaming a current .sql script). Alternatively, you can press and hold [CTRL] + [S].
- In the Save dialog box, enter the appropriate filename. Make sure the extension is .sql or the File type is selected as SQL Script (*.sql). Click Save.
Note: For this course, you need to save your sql scripts in the \\home\\oracle\\labs\\sql1\\labs folder.
- The SQL Worksheet is renamed to the filename that you saved the script as. Make sure you do not enter any other SQL statements in the same worksheet. To continue with other SQL queries, open a new worksheet.

Running Script Files



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Running Script Files

To run the saved .sql script files, perform the following:

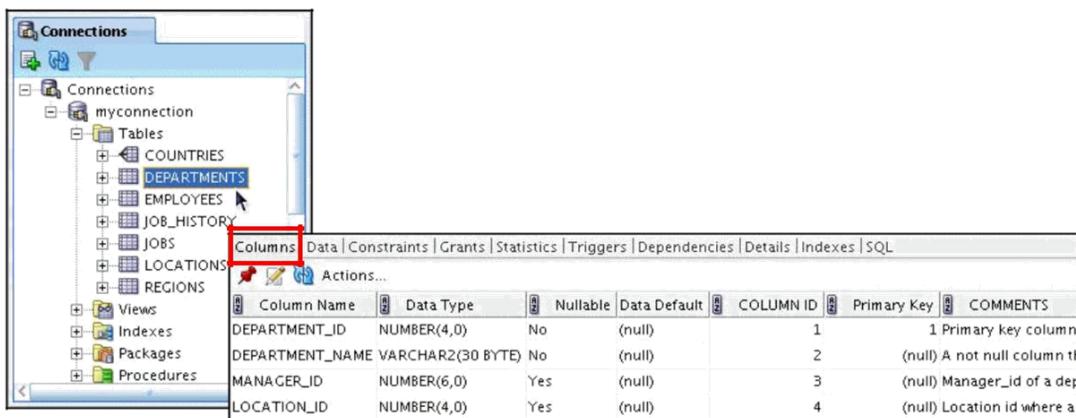
1. From the File menu, select Open. Alternatively, you can press and hold [CTRL] + [O].
2. In the Open dialog box, move to the \home\oracle\labs\sql1\labs folder, or to the location in which you saved the script file, select the file and click Open.
3. The script file opens in a new worksheet. Now, you can run the script by either clicking the Execute Statement icon or the Run Script icon. Again, make sure you do not enter any other SQL statements in the same worksheet. To continue with other SQL queries, open a new worksheet.

Note: You may want to set the default directory to \home\oracle\labs\sql1 folder, so that every time you try to open or save a script, SQL Developer chooses the same path to look for scripts. From Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters. In the right pane, click Browse to set the default path to look for scripts and click OK.

Note: For more details on how to use the Oracle SQL Developer GUI interface for other data objects creation and data retrieval tasks, refer to Appendix E “Using SQL Developer.”

Displaying the Table Structure

- Use the DESCRIBE command to display the structure of a table.
- ```
DESC [RIBE] tablename
```
- Or, select the table in the Connections tree and use the Columns tab to view the table structure.



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Displaying the Table Structure

In SQL Developer, you can display the structure of a table by using the DESCRIBE command. The command displays the column names and the data types, and it shows you whether a column *must* contain data (that is, whether the column has a NOT NULL constraint).

In the syntax, *table name* is the name of any existing table, view, or synonym that is accessible to the user.

Using the SQL Developer GUI interface, you can select the table in the Connections tree and use the Columns tab to view the table structure.

**Note:** The DESCRIBE command is supported by both SQL\*Plus and SQL Developer.

# Using the DESCRIBE Command

**DESCRIBE employees**

```
DESCRIBE employees
Name Null Type
----- -----
EMPLOYEE_ID NOT NULL NUMBER(6)
FIRST_NAME VARCHAR2(20)
LAST_NAME NOT NULL VARCHAR2(25)
EMAIL NOT NULL VARCHAR2(25)
PHONE_NUMBER VARCHAR2(20)
HIRE_DATE NOT NULL DATE
JOB_ID NOT NULL VARCHAR2(10)
SALARY NUMBER(8,2)
COMMISSION_PCT NUMBER(2,2)
MANAGER_ID NUMBER(6)
DEPARTMENT_ID NUMBER(4)

11 rows selected
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Using the DESCRIBE Command

The example in the slide displays information about the structure of the EMPLOYEES table using the DESCRIBE command.

In the resulting display, *Null* indicates that the values for this column may be unknown. *NOT NULL* indicates that a column must contain data. *Type* displays the data type for a column.

The data types are described in the following table:

| Data Type              | Description                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------|
| NUMBER ( <i>p, s</i> ) | Number value having a maximum number of digits <i>p</i> , with <i>s</i> digits to the right of the decimal point |
| VARCHAR2 ( <i>s</i> )  | Variable-length character value of maximum size <i>s</i>                                                         |
| DATE                   | Date and time value between January 1, 4712 B.C. and December 31, A.D. 9999.                                     |
| CHAR ( <i>s</i> )      | Fixed-length character value of size <i>s</i>                                                                    |

# Summary

In this lesson, you should have learned how to:

- Write a SELECT statement that:
  - Returns all rows and columns from a table
  - Returns specified columns from a table
  - Uses column aliases to display more descriptive column headings
- Use the SQL Developer environment to write, save, and execute SQL statements

```
SELECT * | { [DISTINCT] column|expression
[alias],... } FROM table;
```



Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to retrieve data from a database table with the SELECT statement.

```
SELECT * | { [DISTINCT] column [alias],... }
 FROM table;
```

In the syntax:

|                   |                                            |
|-------------------|--------------------------------------------|
| SELECT            | is a list of one or more columns           |
| *                 | selects all columns                        |
| DISTINCT          | suppresses duplicates                      |
| column expression | selects the named column or the expression |
| alias             | gives selected columns different headings  |
| FROM table        | specifies the table containing the columns |

## SQL Developer

SQL Developer is an execution environment that you can use to send SQL statements to the database server and to edit and save SQL statements. Statements can be executed from the SQL prompt or from a script file.

## Practice 1: Overview

This practice covers the following topics:

- Using SQL Developer
- Selecting all data from different tables
- Describing the structure of tables
- Performing arithmetic calculations and specifying column names



Copyright © 2009, Oracle. All rights reserved.

### Practice 1: Overview

This is the first of many practices in this course. The solutions (if you require them) can be found in Appendix A. The practices are intended to cover all the topics that are presented in the corresponding lesson.

*Note the following location for the lab files:*

`\home\oracle\labs\SQL1\labs`

*If you are asked to save any lab files, save them at this location.*

In any practice, there may be exercises that are prefaced with the phrases “If you have time” or “If you want an extra challenge.” Work on these exercises only if you have completed all other exercises in the allocated time and would like a further challenge to your skills.

Perform the practices slowly and precisely. You can experiment with saving and running command files. If you have any questions at any time, ask your instructor.

**Note:** All written practices use Oracle SQL Developer as the development environment. Although it is recommended that you use Oracle SQL Developer, you can also use SQL\*Plus that is available in this course.

## Practice 1

### Part 1

Test your knowledge:

1. The following SELECT statement executes successfully:

```
SELECT last_name, job_id, salary AS Sal
FROM employees;
```

True/False

2. The following SELECT statement executes successfully:

```
SELECT *
FROM job_grades;
```

True/False

3. There are four coding errors in the following statement. Can you identify them?

```
SELECT employee_id, last_name
sal * 12 ANNUAL SALARY
FROM employees;
```

### Part 2

*Note the following location for the lab files:*

\home\oracle\labs\SQL1\labs

*If you are asked to save any lab files, save them at this location.*

To start Oracle SQL Developer, double-click the SQL Developer desktop icon.

Before you begin with the practices, you need a database connection to be able to connect to the database and issue SQL queries.

4. To create a new database connection in the Connections Navigator, right-click Connections.

Select New Connection from the shortcut menu. The New/Select Database Connection dialog box appears.

5. Create a database connection using the following information:

- a. Connection Name: myconnection
- b. Username: ora1
- c. Password: ora1
- d. Hostname: localhost
- e. Port: 1521
- f. SID: ORCL
- g. Ensure that you select the Save Password check box.

## Practice 1 (continued)

You have been hired as a SQL programmer for Acme Corporation. Your first task is to create some reports based on data from the Human Resources tables.

- Your first task is to determine the structure of the DEPARTMENTS table and its contents.

| DESCRIBE departments |          |              |
|----------------------|----------|--------------|
| Name                 | Null     | Type         |
| DEPARTMENT_ID        | NOT NULL | NUMBER(4)    |
| DEPARTMENT_NAME      | NOT NULL | VARCHAR2(30) |
| MANAGER_ID           |          | NUMBER(6)    |
| LOCATION_ID          |          | NUMBER(4)    |
| 4 rows selected      |          |              |

| DEPARTMENT_ID | DEPARTMENT_NAME   | MANAGER_ID | LOCATION_ID |
|---------------|-------------------|------------|-------------|
| 1             | 10 Administration | 200        | 1700        |
| 2             | 20 Marketing      | 201        | 1800        |
| 3             | 50 Shipping       | 124        | 1500        |
| 4             | 60 IT             | 103        | 1400        |
| 5             | 80 Sales          | 149        | 2500        |
| 6             | 90 Executive      | 100        | 1700        |
| 7             | 110 Accounting    | 205        | 1700        |
| 8             | 190 Contracting   | (null)     | 1700        |

- You need to determine the structure of the EMPLOYEES table.

| DESCRIBE employees |          |              |
|--------------------|----------|--------------|
| Name               | Null     | Type         |
| EMPLOYEE_ID        | NOT NULL | NUMBER(6)    |
| FIRST_NAME         |          | VARCHAR2(20) |
| LAST_NAME          | NOT NULL | VARCHAR2(25) |
| EMAIL              | NOT NULL | VARCHAR2(25) |
| PHONE_NUMBER       |          | VARCHAR2(20) |
| HIRE_DATE          | NOT NULL | DATE         |
| JOB_ID             | NOT NULL | VARCHAR2(10) |
| SALARY             |          | NUMBER(8,2)  |
| COMMISSION_PCT     |          | NUMBER(2,2)  |
| MANAGER_ID         |          | NUMBER(6)    |
| DEPARTMENT_ID      |          | NUMBER(4)    |
| 11 rows selected   |          |              |

The HR department wants a query to display the last name, job code, hire date, and employee number for each employee, with the employee number appearing first. Provide an alias STARTDATE for the HIRE\_DATE column. Save your SQL statement to a file named lab\_01\_07.sql so that you can dispatch this file to the HR department.

## Practice 1 (continued)

8. Test your query in the lab\_01\_07.sql file to ensure that it runs correctly.

| #  | EMPLOYEE_ID | LAST_NAME | JOB_ID     | STARTDATE |
|----|-------------|-----------|------------|-----------|
| 1  | 200         | Whalen    | AD_ASST    | 17-SEP-87 |
| 2  | 201         | Hartstein | MK_MAN     | 17-FEB-96 |
| 3  | 202         | Fay       | MK_REP     | 17-AUG-97 |
| 4  | 205         | Higgins   | AC_MGR     | 07-JUN-94 |
| 5  | 206         | Gietz     | AC_ACCOUNT | 07-JUN-94 |
| 6  | 100         | King      | AD_PRES    | 17-JUN-87 |
| 7  | 101         | Kochhar   | AD_VP      | 21-SEP-89 |
| 8  | 102         | De Haan   | AD_VP      | 13-JAN-93 |
| 9  | 103         | Hunold    | IT_PROG    | 03-JAN-90 |
| 10 | 104         | Ernst     | IT_PROG    | 21-MAY-91 |
| 11 | 107         | Lorentz   | IT_PROG    | 07-FEB-99 |
| 12 | 124         | Mourgos   | ST_MAN     | 16-NOV-99 |
| 13 | 141         | Rajs      | ST_CLERK   | 17-OCT-95 |
| 14 | 142         | Davies    | ST_CLERK   | 29-JAN-97 |
| 15 | 143         | Matos     | ST_CLERK   | 15-MAR-98 |
| 16 | 144         | Vargas    | ST_CLERK   | 09-JUL-98 |
| 17 | 149         | Zlotkey   | SA_MAN     | 29-JAN-00 |
| 18 | 174         | Abel      | SA_REP     | 11-MAY-96 |
| 19 | 176         | Taylor    | SA_REP     | 24-MAR-98 |
| 20 | 178         | Grant     | SA_REP     | 24-MAY-99 |

9. The HR department needs a query to display all unique job codes from the EMPLOYEES table.

| #  | JOB_ID     |
|----|------------|
| 1  | AC_ACCOUNT |
| 2  | AC_MGR     |
| 3  | AD_ASST    |
| 4  | AD_PRES    |
| 5  | AD_VP      |
| 6  | IT_PROG    |
| 7  | MK_MAN     |
| 8  | MK_REP     |
| 9  | SA_MAN     |
| 10 | SA_REP     |
| 11 | ST_CLERK   |
| 12 | ST_MAN     |

## Practice 1 (continued)

### Part 3

If you have time, complete the following exercises:

10. The HR department wants more descriptive column headings for its report on employees. Copy the statement from lab\_01\_07.sql to the SQL Developer text box. Name the column headings Emp #, Employee, Job, and Hire Date, respectively. Then run your query again.

|     | Emp # | Employee  | Job        | Hire Date |
|-----|-------|-----------|------------|-----------|
| 1   | 200   | Whalen    | AD_ASST    | 17-SEP-87 |
| 2   | 201   | Hartstein | MK_MAN     | 17-FEB-96 |
| 3   | 202   | Fay       | MK_REP     | 17-AUG-97 |
| 4   | 205   | Higgins   | AC_MGR     | 07-JUN-94 |
| 5   | 206   | Gietz     | AC_ACCOUNT | 07-JUN-94 |
| 6   | 100   | King      | AD_PRES    | 17-JUN-87 |
| 7   | 101   | Kochhar   | AD_VP      | 21-SEP-89 |
| 8   | 102   | De Haan   | AD_VP      | 13-JAN-93 |
| 9   | 103   | Hunold    | IT_PROG    | 03-JAN-90 |
| 10  | 104   | Ernst     | IT_PROG    | 21-MAY-91 |
| ... |       |           |            |           |
| 20  | 178   | Grant     | SA_REP     | 24-MAY-99 |

11. The HR department has requested a report of all employees and their job IDs. Display the last name concatenated with the job ID (separated by a comma and space) and name the column Employee and Title.

|     | Employee and Title |
|-----|--------------------|
| 1   | Abel, SA_REP       |
| 2   | Davies, ST_CLERK   |
| 3   | De Haan, AD_VP     |
| 4   | Ernst, IT_PROG     |
| 5   | Fay, MK_REP        |
| 6   | Gietz, AC_ACCOUNT  |
| 7   | Grant, SA_REP      |
| 8   | Hartstein, MK_MAN  |
| 9   | Higgins, AC_MGR    |
| 10  | Hunold, IT_PROG    |
| ... |                    |
| 20  | Zlotkey, SA_MAN    |

## Practice 1 (continued)

If you want an extra challenge, complete the following exercise:

12. To familiarize yourself with the data in the EMPLOYEES table, create a query to display all the data from that table. Separate each column output with a comma. Name the column THE\_OUTPUT.

| THE_OUTPUT                                                                      |
|---------------------------------------------------------------------------------|
| 1 200,Jennifer,Whalen,JWHALEN,515.123.4444,AD_ASST,101,17-SEP-87,4400,,10       |
| 2 201,Michael,Hartstein,MHARTSTE,515.123.5555,MK_MAN,100,17-FEB-96,13000,,20    |
| 3 202,Pat,Fay,PFAY,603.123.6666,MK_REP,201,17-AUG-97,6000,,20                   |
| 4 205,Shelley,Higgins,SHIGGINS,515.123.8080,AC_MGR,101,07-JUN-94,12000,,110     |
| 5 206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110      |
| 6 100,Steven,King,SKING,515.123.4567,AD_PRES,,17-JUN-87,24000,,90               |
| 7 101,Neena,Kochhar,NKOCHHAR,515.123.4568,AD_VP,100,21-SEP-89,17000,,90         |
| 8 102,Lex,De Haan,LDEHAAN,515.123.4569,AD_VP,100,13-JAN-93,17000,,90            |
| 9 103,Alexander,Hunold,AHUNOLD,590.423.4567,IT_PROG,102,03-JAN-90,9000,,60      |
| 10 104,Bruce,Ernst,BERNST,590.423.4568,IT_PROG,103,21-MAY-91,6000,,60           |
| ...                                                                             |
| 20 178,Kimberely,Grant,KGRANT,011.44.1644.429263,SA_REP,149,24-MAY-99,7000,.15, |

Oracle Internal & Oracle Academy Use Only

# 2

## Restricting and Sorting Data

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

# Objectives

After completing this lesson, you should be able to do the following:

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query
- Use ampersand substitution to restrict and sort output at run time



Copyright © 2009, Oracle. All rights reserved.

## Objectives

When retrieving data from the database, you may need to do the following:

- Restrict the rows of data that are displayed
- Specify the order in which the rows are displayed

This lesson explains the SQL statements that you use to perform these actions.

# Limiting Rows Using a Selection

## EMPLOYEES

|     | EMPLOYEE_ID | LAST_NAME | JOB_ID     | DEPARTMENT_ID |
|-----|-------------|-----------|------------|---------------|
| 1   | 200         | Whalen    | AD_ASST    | 10            |
| 2   | 201         | Hartstein | MK_MAN     | 20            |
| 3   | 202         | Fay       | MK_REP     | 20            |
| 4   | 205         | Higgins   | AC_MGR     | 110           |
| 5   | 206         | Gietz     | AC_ACCOUNT | 110           |
| 6   | 100         | King      | AD_PRES    | 90            |
| 7   | 101         | Kochhar   | AD_VP      | 90            |
| ... |             |           |            |               |
| 20  | 178         | Grant     | SA_REP     | (null)        |

“retrieve all  
employees in  
department 90”

|   | EMPLOYEE_ID | LAST_NAME | JOB_ID  | DEPARTMENT_ID |
|---|-------------|-----------|---------|---------------|
| 1 | 100         | King      | AD_PRES | 90            |
| 2 | 101         | Kochhar   | AD_VP   | 90            |
| 3 | 102         | De Haan   | AD_VP   | 90            |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Limiting Rows Using a Selection

In the example in the slide, assume that you want to display all the employees in department 90. The rows with a value of 90 in the DEPARTMENT\_ID column are the only ones that are returned. This method of restriction is the basis of the WHERE clause in SQL.

## Limiting the Rows That Are Selected

- Restrict the rows that are returned by using the WHERE clause:

```
SELECT * | { [DISTINCT] column|expression [alias],... }
FROM table
[WHERE condition(s)] ;
```

- The WHERE clause follows the FROM clause.



Copyright © 2009, Oracle. All rights reserved.

### Limiting the Rows That Are Selected

You can restrict the rows that are returned from the query by using the WHERE clause. A WHERE clause contains a condition that must be met, and it directly follows the FROM clause. If the condition is true, the row meeting the condition is returned.

In the syntax:

|                           |                                                                                                                                        |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| WHERE<br><i>condition</i> | restricts the query to rows that meet a condition<br>is composed of column names, expressions,<br>constants, and a comparison operator |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------|

The WHERE clause can compare values in columns, literal values, arithmetic expressions, or functions. It consists of three elements:

- Column name
- Comparison condition
- Column name, constant, or list of values

## Using the WHERE Clause

```
SELECT employee_id, last_name, job_id,
department_id FROM employees
WHERE department_id = 90 ;
```

|   | EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|---|-------------|-----------|--------|---------------|
| 1 | 100 King    | AD_PRES   | 90     |               |
| 2 | 101 Kochhar | AD_VP     | 90     |               |
| 3 | 102 De Haan | AD_VP     | 90     |               |



Copyright © 2009, Oracle. All rights reserved.

### Using the WHERE Clause

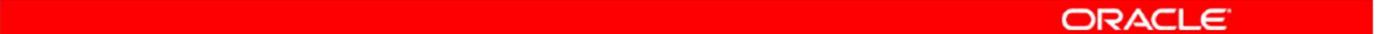
In the example, the SELECT statement retrieves the employee ID, name, job ID, and department number of all employees who are in department 90.

# Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks.
- Character values are case sensitive, and date values are format sensitive.
- The default date format is DD-MON-RR.

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'Whalen';
```

|   | LAST_NAME | JOB_ID  | DEPARTMENT_ID |
|---|-----------|---------|---------------|
| 1 | Whalen    | AD_ASST | 10            |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Character Strings and Dates

Character strings and dates in the WHERE clause must be enclosed in single quotation marks (' '). Number constants, however, should not be enclosed in single quotation marks.

All character searches are case sensitive. In the following example, no rows are returned because the EMPLOYEES table stores all the last names in mixed case:

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'WHALEN';
```

The Oracle Database stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds. The default date display is DD-MON-RR.

**Note:** For details about the RR format and about changing the default date format, see the lesson titled “Using Single-Row Functions to Customize Output.”

# Comparison Conditions

| Operator               | Meaning                        |
|------------------------|--------------------------------|
| =                      | Equal to                       |
| >                      | Greater than                   |
| ≥                      | Greater than or equal to       |
| <                      | Less than                      |
| ≤                      | Less than or equal to          |
| ≠                      | Not equal to                   |
| BETWEEN<br>... AND ... | Between two values (inclusive) |
| IN (set)               | Match any of a list of values  |
| LIKE                   | Match a character pattern      |
| IS NULL                | Is a null value                |

Copyright © 2009, Oracle. All rights reserved.

## Comparison Conditions

Comparison conditions are used in conditions that compare one expression to another value or expression. They are used in the WHERE clause in the following format:

### Syntax

```
... WHERE expr operator value
```

### Example

```
... WHERE hire_date = '01-JAN-95'
... WHERE salary >= 6000
... WHERE last_name = 'Smith'
```

An alias cannot be used in the WHERE clause.

**Note:** The symbols != and ^= can also represent the *not equal to* condition.

# Using Comparison Conditions

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000 ;
```

|   | LAST_NAME | SALARY |
|---|-----------|--------|
| 1 | Matos     | 2600   |
| 2 | Vargas    | 2500   |



Copyright © 2009, Oracle. All rights reserved.

## Using Comparison Conditions

In the example, the SELECT statement retrieves the last name and salary from the EMPLOYEES table for any employee whose salary is less than or equal to \$3,000. Note that there is an explicit value supplied to the WHERE clause. The explicit value of 3000 is compared to the salary value in the SALARY column of the EMPLOYEES table.

## Using the BETWEEN Condition

Use the BETWEEN condition to display rows based on a range of values:

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

Lower limit      Upper limit

|   | LAST_NAME | SALARY |
|---|-----------|--------|
| 1 | Rajs      | 3500   |
| 2 | Davies    | 3100   |
| 3 | Matos     | 2600   |
| 4 | Vargas    | 2500   |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the BETWEEN Condition

You can display rows based on a range of values using the BETWEEN range condition. The range that you specify contains a lower limit and an upper limit.

The SELECT statement in the slide returns rows from the EMPLOYEES table for any employee whose salary is between \$2,500 and \$3,500.

Values that are specified with the BETWEEN condition are inclusive. You must specify the lower limit first.

You can also use the BETWEEN condition on character values:

```
SELECT last_name
FROM employees
WHERE last_name BETWEEN 'King' AND 'Smith';
```

|   | LAST_NAME |
|---|-----------|
| 1 | King      |
| 2 | Kochhar   |
| 3 | Lorentz   |
| 4 | Matos     |
| 5 | Mourgos   |
| 6 | Rajs      |

# Using the IN Condition

Use the `IN` membership condition to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

|   | EMPLOYEE_ID | LAST_NAME | SALARY | MANAGER_ID |
|---|-------------|-----------|--------|------------|
| 1 | 201         | Hartstein | 13000  | 100        |
| 2 | 101         | Kochhar   | 17000  | 100        |
| 3 | 102         | De Haan   | 17000  | 100        |
| 4 | 124         | Moungos   | 5800   | 100        |
| 5 | 149         | Zlotkey   | 10500  | 100        |
| 6 | 200         | Whalen    | 4400   | 101        |
| 7 | 205         | Higgins   | 12000  | 101        |
| 8 | 202         | Fay       | 6000   | 201        |



Copyright © 2009, Oracle. All rights reserved.

## Using the IN Condition

To test for values in a specified set of values, use the `IN` condition. The `IN` condition is also known as the *membership condition*.

The slide example displays employee numbers, last names, salaries, and manager's employee numbers for all the employees whose manager's employee number is 100, 101, or 201.

The `IN` condition can be used with any data type. The following example returns a row from the `EMPLOYEES` table for any employee whose last name is included in the list of names in the `WHERE` clause:

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE last_name IN ('Hartstein', 'Vargas');
```

If characters or dates are used in the list, they must be enclosed in single quotation marks (' ').

## Using the LIKE Condition

- Use the `LIKE` condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
  - `%` denotes zero or many characters.
  - `_` denotes one character.

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%' ;
```

| FIRST_NAME |
|------------|
| Shelley    |
| Steven     |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the LIKE Condition

You may not always know the exact value to search for. You can select rows that match a character pattern by using the `LIKE` condition. The character pattern-matching operation is referred to as a *wildcard* search. Two symbols can be used to construct the search string.

| Symbol         | Description                                        |
|----------------|----------------------------------------------------|
| <code>%</code> | Represents any sequence of zero or more characters |
| <code>_</code> | Represents any single character                    |

The `SELECT` statement in the slide returns the employee first name from the `EMPLOYEES` table for any employee whose first name begins with the letter `S`. Note the uppercase `S`. Names beginning with an `s` are not returned.

The `LIKE` condition can be used as a shortcut for some `BETWEEN` comparisons. The following example displays the last names and hire dates of all employees who joined between January 1995 and December 1995:

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%95' ;
```

## Using the LIKE Condition

- You can combine pattern-matching characters:

```
SELECT last_name
FROM employees
WHERE last_name LIKE '_o%';
```

| LAST_NAME |
|-----------|
| Kochhar   |
| Lorentz   |
| Mourgos   |

- You can use the ESCAPE identifier to search for the actual % and \_ symbols.



Copyright © 2009, Oracle. All rights reserved.

### Combining Wildcard Characters

The % and \_ symbols can be used in any combination with literal characters. The example in the slide displays the names of all employees whose last names have the letter o as the second character.

#### ESCAPE Option

When you need to have an exact match for the actual % and \_ characters, use the ESCAPE option. This option specifies what the escape character is. If you want to search for strings that contain SA\_, you can use the following SQL statement:

```
SELECT employee_id, last_name, job_id
FROM employees WHERE job_id LIKE '%SA_%' ESCAPE '\';
```

| EMPLOYEE_ID | LAST_NAME | JOB_ID |
|-------------|-----------|--------|
| 1           | Zlotkey   | SA_MAN |
| 2           | Abel      | SA_REP |
| 3           | Taylor    | SA_REP |
| 4           | Grant     | SA_REP |

The ESCAPE option identifies the backslash (\) as the escape character. In the pattern, the escape character precedes the underscore (\_). This causes the Oracle Server to interpret the underscore literally.

# Using the NULL Conditions

Test for nulls with the `IS NULL` operator.

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

| LAST_NAME | MANAGER_ID |
|-----------|------------|
| King      | (null)     |

Copyright © 2009, Oracle. All rights reserved.

## Using the NULL Conditions

The NULL conditions include the `IS NULL` condition and the `IS NOT NULL` condition.

The `IS NULL` condition tests for nulls. A null value means the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with `=` because a null cannot be equal or unequal to any value. The slide example retrieves the last names and managers of all employees who do not have a manager.

Here is another example: To display last name, job ID, and commission for all employees who are *not* entitled to receive a commission, use the following SQL statement:

```
SELECT last_name, job_id, commission_pct
FROM employees
WHERE commission_pct IS NULL;
```

| LAST_NAME | JOB_ID   | COMMISSION_PCT |
|-----------|----------|----------------|
| Whalen    | AD_ASST  | (null)         |
| Hartstein | MK_MAN   | (null)         |
| Fay       | MK_REP   | (null)         |
| Higgins   | AC_MGR   | (null)         |
| ...       |          |                |
| 16 Vargas | ST_CLERK | (null)         |

# Logical Conditions

| Operator | Meaning                                                   |
|----------|-----------------------------------------------------------|
| AND      | Returns TRUE if <i>both</i> component conditions are true |
| OR       | Returns TRUE if <i>either</i> component condition is true |
| NOT      | Returns TRUE if the following condition is false          |



Copyright © 2009, Oracle. All rights reserved.

## Logical Conditions

A logical condition combines the result of two component conditions to produce a single result based on those conditions, or it inverts the result of a single condition. A row is returned only if the overall result of the condition is true.

Three logical operators are available in SQL:

- AND
- OR
- NOT

All the examples so far have specified only one condition in the WHERE clause. You can use several conditions in one WHERE clause using the AND and OR operators.

# Using the AND Operator

AND requires both conditions to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >=10000
AND job_id LIKE '%MAN%' ;
```

|   | EMPLOYEE_ID | LAST_NAME | JOB_ID | SALARY |
|---|-------------|-----------|--------|--------|
| 1 | 201         | Hartstein | MK_MAN | 13000  |
| 2 | 149         | Zlotkey   | SA_MAN | 10500  |

Copyright © 2009, Oracle. All rights reserved.

## Using the AND Operator

In the example, both conditions must be true for any record to be selected. Therefore, only employees who have a job title that contains the string ‘MAN’ *and* earn \$10,000 or more are selected.

All character searches are case sensitive. No rows are returned if ‘MAN’ is not uppercase. Character strings must be enclosed in quotation marks.

### AND Truth Table

The following table shows the results of combining two expressions with AND:

| AND   | TRUE  | FALSE | NULL  |
|-------|-------|-------|-------|
| TRUE  | TRUE  | FALSE | NULL  |
| FALSE | FALSE | FALSE | FALSE |
| NULL  | NULL  | FALSE | NULL  |

# Using the OR Operator

OR requires either condition to be true:

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%' ;
```

|   | EMPLOYEE_ID   | LAST_NAME | JOB_ID | SALARY |
|---|---------------|-----------|--------|--------|
| 1 | 201 Hartstein | MK_MAN    | 13000  |        |
| 2 | 205 Higgins   | AC_MGR    | 12000  |        |
| 3 | 100 King      | AD_PRES   | 24000  |        |
| 4 | 101 Kochhar   | AD_VP     | 17000  |        |
| 5 | 102 De Haan   | AD_VP     | 17000  |        |
| 6 | 124 Mourgos   | ST_MAN    | 5800   |        |
| 7 | 149 Zlotkey   | SA_MAN    | 10500  |        |
| 8 | 174 Abel      | SA_REP    | 11000  |        |

Copyright © 2009, Oracle. All rights reserved.

## Using the OR Operator

In the example, either condition can be true for any record to be selected. Therefore, any employee who has a job ID that contains the string ‘MAN’ or earns \$10,000 or more is selected.

### OR Truth Table

The following table shows the results of combining two expressions with OR:

| OR    | TRUE | FALSE | NULL |
|-------|------|-------|------|
| TRUE  | TRUE | TRUE  | TRUE |
| FALSE | TRUE | FALSE | NULL |
| NULL  | TRUE | NULL  | NULL |

# Using the NOT Operator

```
SELECT last_name, job_id
FROM employees
WHERE job_id
 NOT IN ('IT_PROG', 'ST_CLERK', 'SA REP') ;
```

|    | LAST_NAME | JOB_ID     |
|----|-----------|------------|
| 1  | De Haan   | AD_VP      |
| 2  | Fay       | MK_REP     |
| 3  | Gietz     | AC_ACCOUNT |
| 4  | Hartstein | MK_MAN     |
| 5  | Higgins   | AC_MGR     |
| 6  | King      | AD_PRES    |
| 7  | Kochhar   | AD_VP      |
| 8  | Mourgos   | ST_MAN     |
| 9  | Whalen    | AD_ASST    |
| 10 | Zlotkey   | SA_MAN     |



Copyright © 2009, Oracle. All rights reserved.

## Using the NOT Operator

The slide example displays the last name and job ID of all employees whose job ID is *not* IT\_PROG, ST\_CLERK, or SA REP.

### NOT Truth Table

The following table shows the result of applying the NOT operator to a condition:

| NOT | TRUE  | FALSE | NULL |
|-----|-------|-------|------|
|     | FALSE | TRUE  | NULL |

**Note:** The NOT operator can also be used with other SQL operators, such as BETWEEN, LIKE, and NULL.

```
... WHERE job_id NOT IN ('AC_ACCOUNT', 'AD_VP')
... WHERE salary NOT BETWEEN 10000 AND 15000
... WHERE last_name NOT LIKE '%A%'
... WHERE commission_pct IS NOT NULL
```

# Rules of Precedence

| Operator | Meaning                       |
|----------|-------------------------------|
| 1        | Arithmetic operators          |
| 2        | Concatenation operator        |
| 3        | Comparison conditions         |
| 4        | IS [NOT] NULL, LIKE, [NOT] IN |
| 5        | [NOT] BETWEEN                 |
| 6        | Not equal to                  |
| 7        | NOT logical condition         |
| 8        | AND logical condition         |
| 9        | OR logical condition          |

You can use parentheses to override rules of precedence.



Copyright © 2009, Oracle. All rights reserved.

## Rules of Precedence

The rules of precedence determine the order in which expressions are evaluated and calculated. The table lists the default order of precedence. You can override the default order by using parentheses around the expressions that you want to calculate first.

## Rules of Precedence

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

1

|   | LAST_NAME | JOB_ID  | SALARY |
|---|-----------|---------|--------|
| 1 | King      | AD_PRES | 24000  |
| 2 | Abel      | SA_REP  | 11000  |
| 3 | Taylor    | SA_REP  | 8600   |
| 4 | Grant     | SA_REP  | 7000   |

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

2

|   | LAST_NAME | JOB_ID  | SALARY |
|---|-----------|---------|--------|
| 1 | King      | AD_PRES | 24000  |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Rules of Precedence (continued)

#### 1. Example of the Precedence of the AND Operator

In this example, there are two conditions:

- The first condition is that the job ID is AD\_PRES *and* the salary is greater than \$15,000.
- The second condition is that the job ID is SA\_REP.

Therefore, the SELECT statement reads as follows:

“Select the row if an employee is a president *and* earns more than \$15,000, *or* if the employee is a sales representative.”

#### 2. Example of Using Parentheses

In this example, there are two conditions:

- The first condition is that the job ID is AD\_PRES *or* SA\_REP.
- The second condition is that salary is greater than \$15,000.

Therefore, the SELECT statement reads as follows:

“Select the row if an employee is a president *or* a sales representative, *and* if the employee earns more than \$15,000.”

## Using the ORDER BY Clause

- Sort retrieved rows with the ORDER BY clause:
  - ASC: ascending order, default
  - DESC: descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date ;
```

|     | LAST_NAME | JOB_ID  | DEPARTMENT_ID | HIRE_DATE |
|-----|-----------|---------|---------------|-----------|
| 1   | King      | AD_PRES | 90            | 17-JUN-87 |
| 2   | Whalen    | AD_ASST | 10            | 17-SEP-87 |
| 3   | Kochhar   | AD_VP   | 90            | 21-SEP-89 |
| 4   | Hunold    | IT_PROG | 60            | 03-JAN-90 |
| ... |           |         |               |           |
| 20  | Zlotkey   | SA_MAN  | 80            | 29-JAN-00 |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the ORDER BY Clause

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. If you use the ORDER BY clause, it must be the last clause of the SQL statement. You can specify an expression, an alias, or a column position as the sort condition.

### Syntax

```
SELECT expr
FROM table
[WHERE condition(s)]
[ORDER BY {column, expr, numeric_position} [ASC|DESC]] ;
```

In the syntax:

|          |                                                                |
|----------|----------------------------------------------------------------|
| ORDER BY | specifies the order in which the retrieved rows are displayed  |
| ASC      | orders the rows in ascending order (this is the default order) |
| DESC     | orders the rows in descending order                            |

If the ORDER BY clause is not used, the sort order is undefined, and the Oracle server may not fetch rows in the same order for the same query twice. Use the ORDER BY clause to display the rows in a specific order.

# Sorting

- Sorting in descending order:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire date DESC ;
```

1

- Sorting by column alias:

```
SELECT employee_id, last_name, salary*12 annsal
FROM employees
ORDER BY annsal ;
```

2

- Sorting by multiple columns:

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY department_id, salary DESC;
```

3

Copyright © 2009, Oracle. All rights reserved.

## Default Ordering of Data

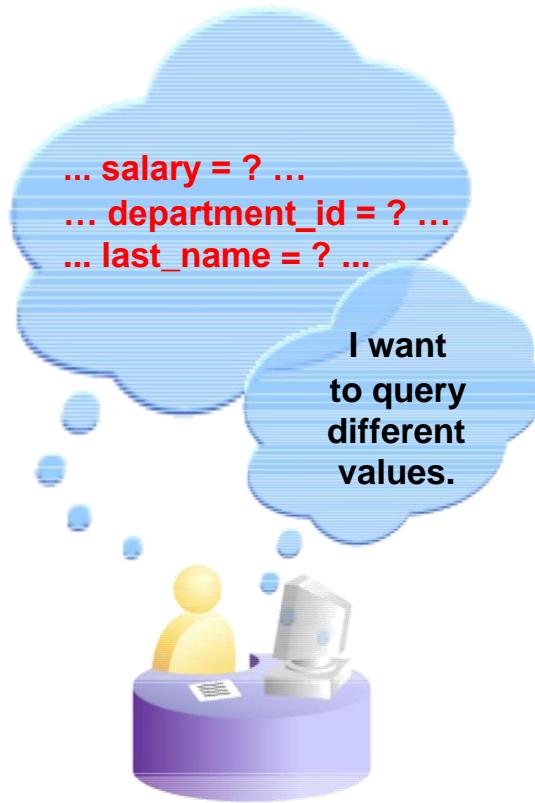
The default sort order is ascending:

- Numeric values are displayed with the lowest values first (for example, 1 to 999).
- Date values are displayed with the earliest value first (for example, 01-JAN-92 before 01-JAN-95).
- Character values are displayed in alphabetical order (for example, A first and Z last).
- Null values are displayed last for ascending sequences and first for descending sequences.
- You can sort by a column that is not in the SELECT list.

## Examples

- To reverse the order in which rows are displayed, specify the DESC keyword after the column name in the ORDER BY clause. The slide example sorts the result by the most recently hired employee.
- You can use a column alias in the ORDER BY clause. The slide example sorts the data by annual salary.
- You can sort query results by more than one column. The sort limit is the number of columns in the given table. In the ORDER BY clause, specify the columns and separate the column names using commas. If you want to reverse the order of a column, specify DESC after its name.

# Substitution Variables



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Substitution Variables

The examples so far have been hard-coded. In a finished application, the user would trigger the report, and the report would run without further prompting. The range of data would be predetermined by the fixed WHERE clause in the script file.

Using SQL Developer, you can create reports that prompt users to supply their own values to restrict the range of data returned by using substitution variables. You can embed *substitution variables* in a command file or in a single SQL statement. A variable can be thought of as a container in which the values are temporarily stored. When the statement is run, the value is substituted.

# Substitution Variables

- Use substitution variables to:
  - Temporarily store values with single-ampersand (&) and double-ampersand (&&) substitution
- Use substitution variables to supplement the following:
  - WHERE conditions
  - ORDER BY clauses
  - Column expressions
  - Table names
  - Entire SELECT statements



Copyright © 2009, Oracle. All rights reserved.

## Substitution Variables (continued)

You can use single-ampersand (&) substitution variables to temporarily store values.

You can predefine variables in by using the `DEFINE` command. `DEFINE` creates and assigns a value to a variable.

### Examples of Restricted Ranges of Data

- Reporting figures only for the current quarter or specified date range
- Reporting on data relevant only to the user requesting the report
- Displaying personnel only within a given department

### Other Interactive Effects

Interactive effects are not restricted to direct user interaction with the `WHERE` clause. The same principles can be used to achieve other goals, such as:

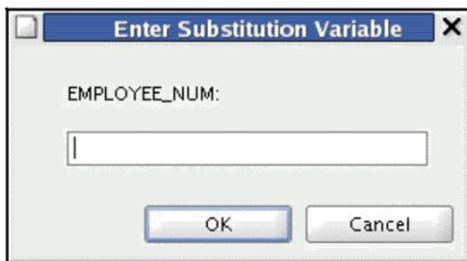
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

**Note:** Both SQL Developer and SQL\* Plus support the substitution variables and the `DEFINE/UNDEFINE` commands. However SQL Developer and SQL\* Plus do not support validation checks (except for data type) on user input.

# Using the & Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for a value:

```
SELECT employee_id, last_name, salary,
department_id FROM employees
WHERE employee_id = &employee_num ;
```



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

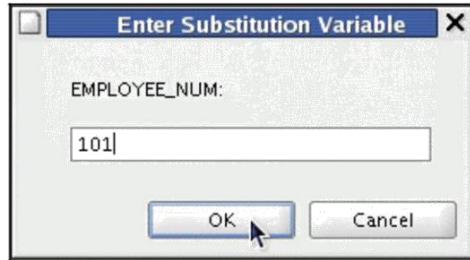
## Single-Ampersand Substitution Variable

When running a report, users often want to restrict the data that is returned dynamically. SQL\*Plus or SQL Developer provides this flexibility with user variables. Use an ampersand (&) to identify each variable in your SQL statement. You do not need to define the value of each variable.

| Notation                        | Description                                                                                                                                                                       |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&amp;user_variable</code> | Indicates a variable in a SQL statement; if the variable does not exist, SQL*Plus or SQL Developer prompts the user for a value (the new variable is discarded after it is used.) |

The example in the slide creates a SQL Developer substitution variable for an employee number. When the statement is executed, SQL Developer prompts the user for an employee number and then displays the employee number, last name, salary, and department number for that employee. With the single ampersand, the user is prompted every time the command is executed, if the variable does not exist.

# Using the & Substitution Variable



|   | EMPLOYEE_ID | LAST_NAME | SALARY | DEPARTMENT_ID |
|---|-------------|-----------|--------|---------------|
| 1 | 101         | Kochhar   | 17000  | 90            |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Single-Ampersand Substitution Variable (continued)

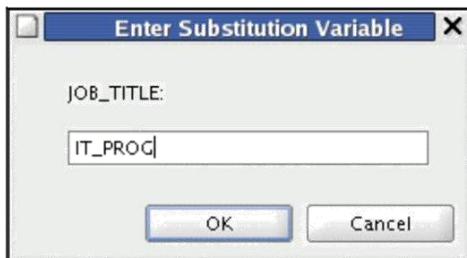
When SQL Developer detects that the SQL statement contains an ampersand, you are prompted to enter a value for the substitution variable that is named in the SQL statement.

After you enter a value and click the OK button, the results are displayed in the Results tab of the SQL Developer session.

# Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:

```
SELECT last_name, department_id, salary*12
FROM employees
WHERE job_id = '&job_title' ;
```



| LAST_NAME | DEPARTMENT_ID | SALARY*12 |
|-----------|---------------|-----------|
| 1 Hunold  | 60            | 108000    |
| 2 Ernst   | 60            | 72000     |
| 3 Lorentz | 60            | 50400     |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Specifying Character and Date Values with Substitution Variables

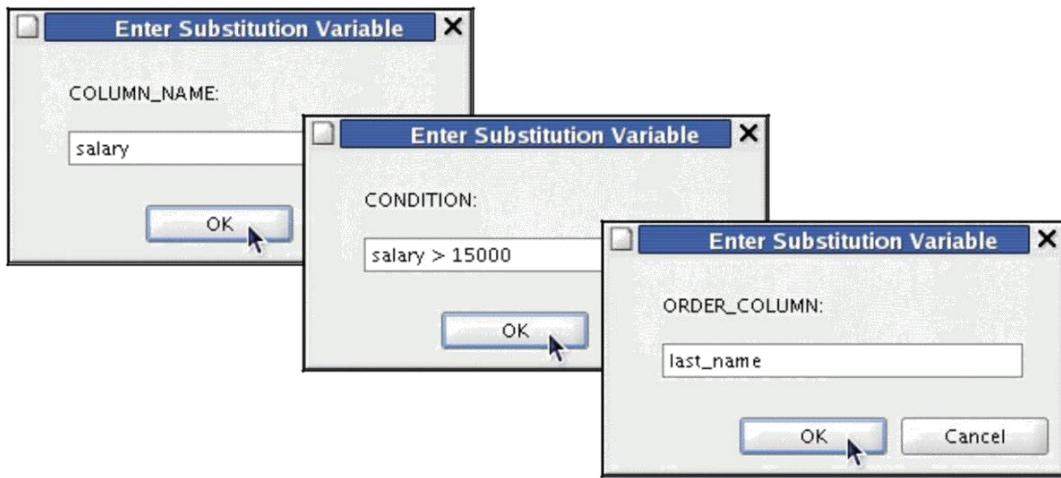
In a WHERE clause, date and character values must be enclosed in single quotation marks. The same rule applies to the substitution variables.

Enclose the variable in single quotation marks within the SQL statement itself.

The slide shows a query to retrieve the employee names, department numbers, and annual salaries of all employees based on the job title value of the SQL Developer substitution variable.

# Specifying Column Names, Expressions, and Text

```
SELECT employee_id, last_name, job_id, &column_name
FROM employees
WHERE &condition
ORDER BY &order_column ;
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Specifying Column Names, Expressions, and Text

Not only can you use the substitution variables in the WHERE clause of a SQL statement, but these variables can also be used to substitute for column names, expressions, or text.

### Example

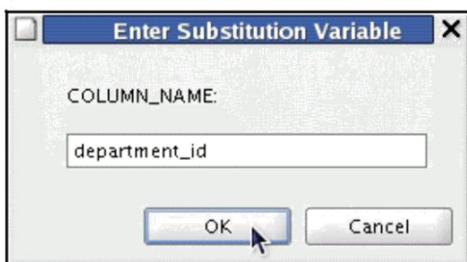
The slide example displays the employee number, name, job title, and any other column that is specified by the user at run time, from the EMPLOYEES table. For each substitution variable in the SELECT statement, you are prompted to enter a value, and you then click the OK button to proceed. If you do not enter a value for the substitution variable, you get an error when you execute the preceding statement.

**Note:** A substitution variable can be used anywhere in the SELECT statement, except as the first word entered at the command prompt.

# Using the && Substitution Variable

Use the double ampersand (&&) if you want to reuse the variable value without prompting the user each time:

```
SELECT employee_id, last_name, job_id, &&column_name
FROM employees
ORDER BY &&column_name ;
```



|     | EMPLOYEE_ID | LAST_NAME | JOB_ID  | DEPARTMENT_ID |
|-----|-------------|-----------|---------|---------------|
| 1   | 200         | Whalen    | AD_ASST | 10            |
| ... |             |           |         |               |
| 20  | 178         | Grant     | SA_REP  | (null)        |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Double-Ampersand Substitution Variable

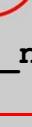
You can use the double-ampersand (&&) substitution variable if you want to reuse the variable value without prompting the user each time. The user sees the prompt for the value only once. In the example in the slide, the user is asked to give the value for variable column\_name only once. The value that is supplied by the user (department\_id) is used for both display and ordering of data. SQL Developer stores the value that is supplied by using the DEFINE command; it uses it again whenever you reference the variable name. After a user variable is in place, you need to use the UNDEFINE command to delete it as follows:

```
UNDEFINE column_name
```

## Using the DEFINE Command

- Use the DEFINE command to create and assign a value to a variable.
- Use the UNDEFINE command to remove a variable.

```
DEFINE employee_num = 200
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num ;
UNDEFINE employee_num
```



Copyright © 2009, Oracle. All rights reserved.

### Using the DEFINE Command

The example shown creates a substitution variable for an employee number by using the DEFINE command. At run time, this displays the employee number, name, salary, and department number for that employee.

Because the variable is created using the SQL Developer DEFINE command, the user is not prompted to enter a value for the employee number. Instead, the defined variable value is automatically substituted in the SELECT statement.

The EMPLOYEE\_NUM substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

## Using the VERIFY Command

Use the VERIFY command to toggle the display of the substitution variable, both before and after SQL Developer replaces substitution variables with values:

The screenshot shows the Oracle SQL Developer interface. In the SQL Worksheet, a code block is shown with the first line 'SET VERIFY ON' highlighted by a red box. The subsequent SQL query uses a substitution variable &employee\_num. In the 'Script Output' tab, the command is run again, but this time it shows the original text of the command including the substitution variable. The output tab also displays the results of the query, which is a single row for employee ID 200 with last name Whalen and salary 4400.

```

SET VERIFY ON
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id = &employee_num;

```

| EMPLOYEE_ID | LAST_NAME | SALARY |
|-------------|-----------|--------|
| 200         | Whalen    | 4400   |

1 rows selected

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Using the VERIFY Command

To confirm the changes in the SQL statement, use the VERIFY command. Setting SET VERIFY ON forces SQL Developer to display the text of a command after it replaces substitution variables with values. To see the VERIFY output, you should use the Run Script (F5) icon in the SQL Worksheet. SQL Developer displays the text of a command after it replaces substitution variables with values, in the Script Output tab as shown in the slide.

The example in the slide displays the new value of the EMPLOYEE\_ID column in the SQL statement followed by the output.

### SQL\*Plus System Variables

SQL\*Plus uses various system variables that control the working environment. One of the variables is VERIFY. To obtain a complete list of all the system variables, you can issue the SHOW ALL command on the SQL\*Plus command prompt.

# Summary

In this lesson, you should have learned how to:

- Use the WHERE clause to restrict rows of output:
  - Use the comparison conditions
  - Use the BETWEEN, IN, LIKE, and NULL conditions
  - Apply the logical AND, OR, and NOT operators
- Use the ORDER BY clause to sort rows of output:

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table
[WHERE condition(s)]
[ORDER BY {column, expr, alias} [ASC|DESC]] ;
```

- Use ampersand substitution to restrict and sort output at run time



Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned about restricting and sorting rows that are returned by the SELECT statement. You should also have learned how to implement various operators and conditions.

By using the substitution variables, you can add flexibility to your SQL statements. You can query users at run time and enable them to specify criteria.

## Practice 2: Overview

This practice covers the following topics:

- Selecting data and changing the order of the rows that are displayed
- Restricting rows by using the WHERE clause
- Sorting rows by using the ORDER BY clause
- Using substitution variables to add flexibility to your SQL SELECT statements



Copyright © 2009, Oracle. All rights reserved.

### Practice 2: Overview

In this practice, you build more reports, including statements that use the WHERE clause and the ORDER BY clause. You make the SQL statements more reusable and generic by including ampersand substitution.

## Practice 2

The HR department needs your assistance with creating some queries.

- Because of budget issues, the HR department needs a report that displays the last name and salary of employees who earn more than \$12,000. Place your SQL statement in a text file named lab\_02\_01.sql. Run your query.

|   | LAST_NAME | SALARY |
|---|-----------|--------|
| 1 | Hartstein | 13000  |
| 2 | King      | 24000  |
| 3 | Kochhar   | 17000  |
| 4 | De Haan   | 17000  |

- Create a report that displays the last name and department number for employee 176.

|   | LAST_NAME | DEPARTMENT_ID |
|---|-----------|---------------|
| 1 | Taylor    | 80            |

- The HR department needs to find high-salary and low-salary employees. Modify lab\_02\_01.sql to display the last name and salary for any employee whose salary is not in the \$5,000–\$12,000 range. Place your SQL statement in a text file named lab\_02\_03.sql.

|    | LAST_NAME | SALARY |
|----|-----------|--------|
| 1  | Whalen    | 4400   |
| 2  | Hartstein | 13000  |
| 3  | King      | 24000  |
| 4  | Kochhar   | 17000  |
| 5  | De Haan   | 17000  |
| 6  | Lorentz   | 4200   |
| 7  | Rajs      | 3500   |
| 8  | Davies    | 3100   |
| 9  | Matos     | 2600   |
| 10 | Vargas    | 2500   |

- Create a report to display the last name, job ID, and start date for the employees whose last names are Matos and Taylor. Order the query in ascending order by start date.

|   | LAST_NAME | JOB_ID   | HIRE_DATE |
|---|-----------|----------|-----------|
| 1 | Matos     | ST_CLERK | 15-MAR-98 |
| 2 | Taylor    | SA_REP   | 24-MAR-98 |

## Practice 2 (continued)

5. Display the last name and department number of all employees in departments 20 or 50 in ascending alphabetical order by name.

|   | LAST_NAME | DEPARTMENT_ID |
|---|-----------|---------------|
| 1 | Davies    | 50            |
| 2 | Fay       | 20            |
| 3 | Hartstein | 20            |
| 4 | Matos     | 50            |
| 5 | Mourgos   | 50            |
| 6 | Rajs      | 50            |
| 7 | Vargas    | 50            |

6. Modify lab\_02\_03.sql to display the last name and salary of employees who earn between \$5,000 and \$12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary, respectively. Resave lab\_02\_03.sql as lab\_02\_06.sql. Run the statement in lab\_02\_06.sql.

|   | Employee | Monthly Salary |
|---|----------|----------------|
| 1 | Fay      | 6000           |
| 2 | Mourgos  | 5800           |

7. The HR department needs a report that displays the last name and hire date for all employees who were hired in 1994.

|   | LAST_NAME | HIRE_DATE |
|---|-----------|-----------|
| 1 | Higgins   | 07-JUN-94 |
| 2 | Gietz     | 07-JUN-94 |

8. Create a report to display the last name and job title of all employees who do not have a manager.

|   | LAST_NAME | JOB_ID  |
|---|-----------|---------|
| 1 | King      | AD_PRES |

9. Create a report to display the last name, salary, and commission of all employees who earn commissions. Sort the data in descending order of salary and commissions.

|   | LAST_NAME | SALARY | COMMISSION_PCT |
|---|-----------|--------|----------------|
| 1 | Abel      | 11000  | 0.3            |
| 2 | Zlotkey   | 10500  | 0.2            |
| 3 | Taylor    | 8600   | 0.2            |
| 4 | Grant     | 7000   | 0.15           |

## Practice 2 (continued)

10. Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary of employees who earn more than an amount that the user specifies after a prompt. (You can use the query that you created in practice exercise 1 and modify it.) Save this query to a file named `lab_02_10.sql`. If you enter `12000` when prompted, the report displays the following results:

|   | LAST_NAME | SALARY |
|---|-----------|--------|
| 1 | Hartstein | 13000  |
| 2 | King      | 24000  |
| 3 | Kochhar   | 17000  |
| 4 | De Haan   | 17000  |

11. The HR department wants to run reports based on a manager. Create a query that prompts the user for a manager ID and generates the employee ID, last name, salary, and department for that manager's employees. The HR department wants the ability to sort the report on a selected column. You can test the data with the following values:

manager ID = 103, sorted by employee last name:

|   | EMPLOYEE_ID | LAST_NAME | SALARY | DEPARTMENT_ID |
|---|-------------|-----------|--------|---------------|
| 1 | 104         | Ernst     | 6000   | 60            |
| 2 | 107         | Lorentz   | 4200   | 60            |

manager ID = 201, sorted by salary:

|   | EMPLOYEE_ID | LAST_NAME | SALARY | DEPARTMENT_ID |
|---|-------------|-----------|--------|---------------|
| 1 | 202         | Fay       | 6000   | 20            |

manager ID = 124, sorted by employee ID:

|   | EMPLOYEE_ID | LAST_NAME | SALARY | DEPARTMENT_ID |
|---|-------------|-----------|--------|---------------|
| 1 | 141         | Rajs      | 3500   | 50            |
| 2 | 142         | Davies    | 3100   | 50            |
| 3 | 143         | Matos     | 2600   | 50            |
| 4 | 144         | Vargas    | 2500   | 50            |

## Practice 2 (continued)

If you have time, complete the following exercises:

12. Display all employee last names in which the third letter of the name is “a.”

|   | LAST_NAME |
|---|-----------|
| 1 | Grant     |
| 2 | Whalen    |

13. Display the last names of all employees who have both an *a* and an *e* in their last name.

|   | LAST_NAME |
|---|-----------|
| 1 | Davies    |
| 2 | De Haan   |
| 3 | Hartstein |
| 4 | Whalen    |

If you want an extra challenge, complete the following exercises:

14. Display the last name, job, and salary for all employees whose jobs are either that of a sales representative or a stock clerk, and whose salaries are not equal to \$2,500, \$3,500, or \$7,000.

|   | LAST_NAME | JOB_ID   | SALARY |
|---|-----------|----------|--------|
| 1 | Abel      | SA_REP   | 11000  |
| 2 | Taylor    | SA_REP   | 8600   |
| 3 | Davies    | ST_CLERK | 3100   |
| 4 | Matos     | ST_CLERK | 2600   |

15. Modify lab\_02\_06.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab\_02\_06.sql as lab\_02\_15.sql. Rerun the statement in lab\_02\_15.sql.

|   | Employee | Monthly Salary | COMMISSION_PCT |
|---|----------|----------------|----------------|
| 1 | Zlotkey  | 10500          | 0.2            |
| 2 | Taylor   | 8600           | 0.2            |

# Using Single-Row Functions to Customize Output

Copyright © 2009, Oracle. All rights reserved.

ORACLE®

# Objectives

After completing this lesson, you should be able to do the following:

- Describe various types of functions that are available in SQL
- Use character, number, and date functions in SELECT statements
- Describe the use of conversion functions

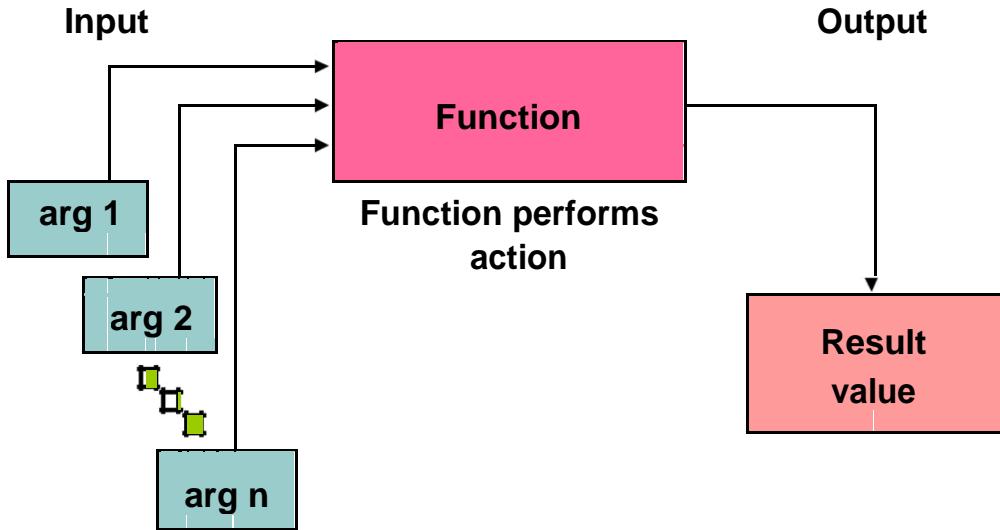


Copyright © 2009, Oracle. All rights reserved.

## Objectives

Functions make the basic query block more powerful, and they are used to manipulate data values. This is the first of two lessons that explore functions. It focuses on single-row character, number, and date functions, as well as those functions that convert data from one type to another (for example, conversion from character data to numeric data).

# SQL Functions



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## SQL Functions

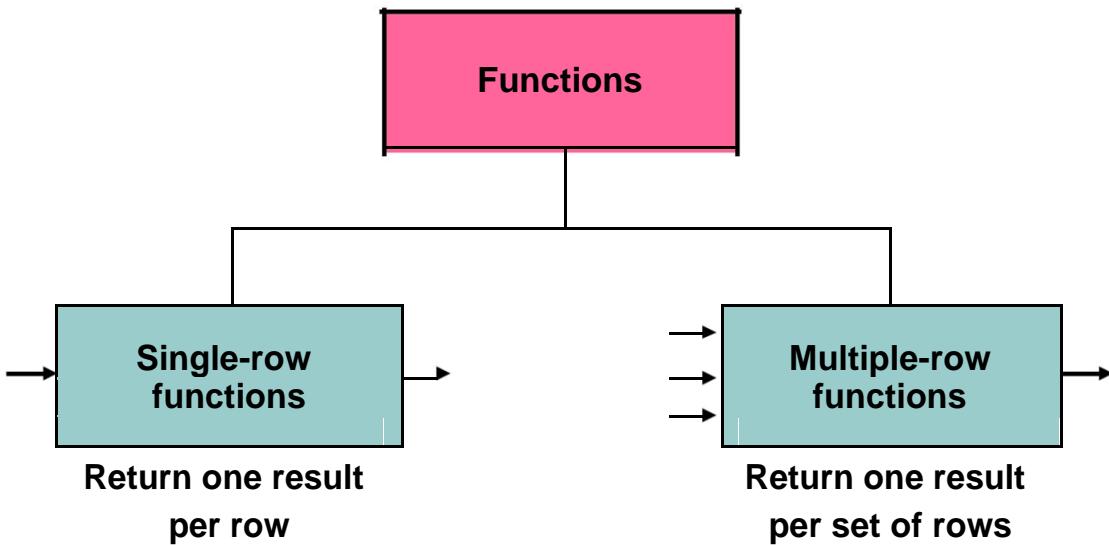
Functions are a very powerful feature of SQL. They can be used to do the following:

- Perform calculations on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

SQL functions sometimes take arguments and always return a value.

**Note:** Most of the functions that are described in this lesson are specific to the Oracle version of SQL.

## Two Types of SQL Functions



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL Functions (continued)

There are two types of functions:

- Single-row functions
- Multiple-row functions

#### Single-Row Functions

These functions operate on single rows only and return one result per row. There are different types of single-row functions. This lesson covers the following ones:

- Character
- Number
- Date
- Conversion
- General

#### Multiple-Row Functions

Functions can manipulate groups of rows to give one result per group of rows. These functions are also known as *group functions* (covered in lesson 4).

**Note:** For more information and a complete list of available functions and their syntax, see *Oracle SQL Reference*.

# Single-Row Functions

Single-row functions:

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

```
function_name [(arg1, arg2, . . .)]
```

Copyright © 2009, Oracle. All rights reserved.

## Single-Row Functions

Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row that is returned by the query. An argument can be one of the following:

- User-supplied constant
- Variable value
- Column name
- Expression

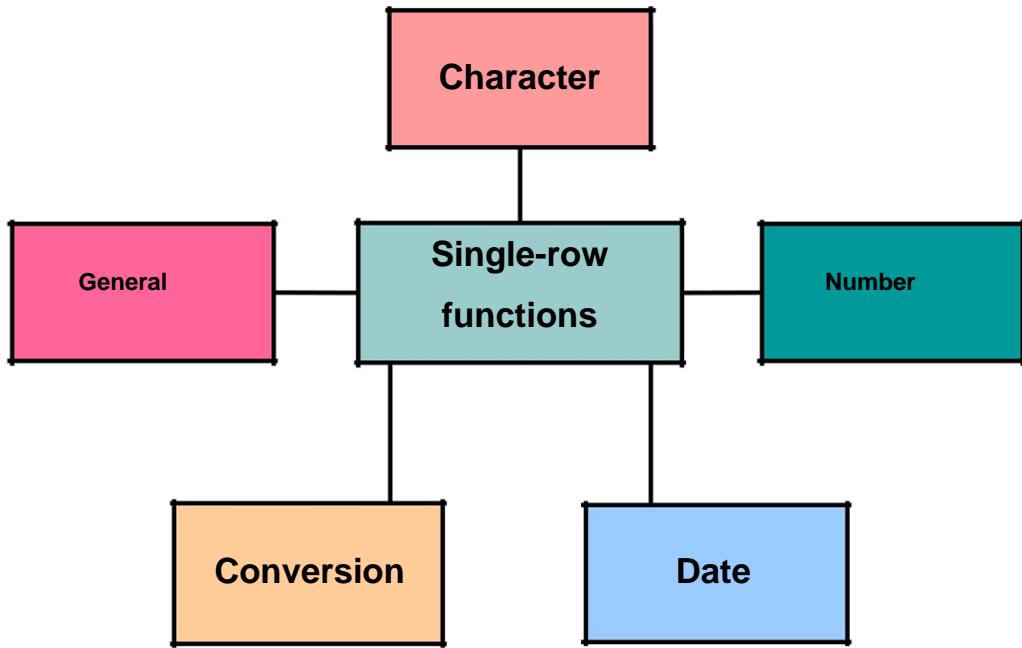
Features of single-row functions include:

- Acting on each row that is returned in the query
- Returning one result per row
- Possibly returning a data value of a different type than the one that is referenced
- Possibly expecting one or more arguments
- Can be used in SELECT, WHERE, and ORDER BY clauses; can be nested

In the syntax:

|                      |                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------|
| <i>function_name</i> | is the name of the function                                                                         |
| <i>arg1, arg2</i>    | is any argument to be used by the function. This can be represented by a column name or expression. |

# Single-Row Functions



ORACLE®

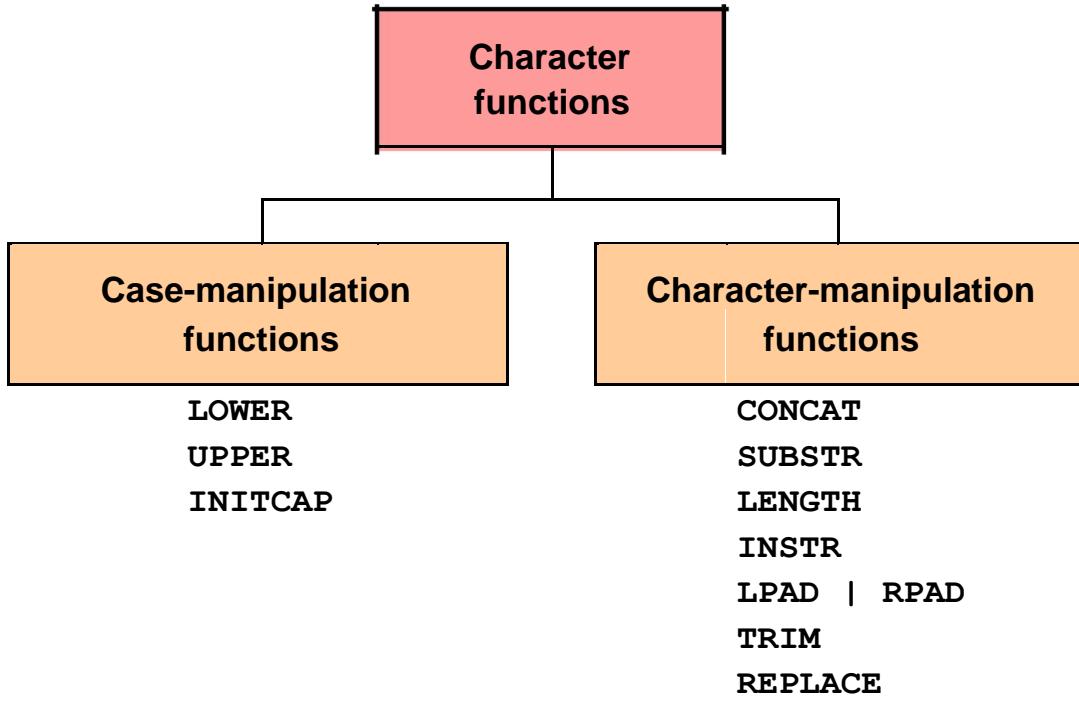
Copyright © 2009, Oracle. All rights reserved.

## Single-Row Functions (continued)

This lesson covers the following single-row functions:

- **Character functions:** Accept character input and can return both character and number values
- **Number functions:** Accept numeric input and return numeric values
- **Date functions:** Operate on values of the DATE data type (All date functions return a value of DATE data type except the MONTHS\_BETWEEN function, which returns a number.)
- **Conversion functions:** Convert a value from one data type to another
- **General functions:**
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
  - CASE
  - DECODE

# Character Functions



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Character Functions

Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-manipulation functions
- Character-manipulation functions

| Function                                                   | Purpose                                                                                                                                                                                                                                                                             |
|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOWER ( <i>column expression</i> )                         | Converts alpha character values to lowercase                                                                                                                                                                                                                                        |
| UPPER ( <i>column expression</i> )                         | Converts alpha character values to uppercase                                                                                                                                                                                                                                        |
| INITCAP ( <i>column expression</i> )                       | Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase                                                                                                                                                                      |
| CONCAT ( <i>column1 expression1, column2 expression2</i> ) | Concatenates the first character value to the second character value; equivalent to concatenation operator (  )                                                                                                                                                                     |
| SUBSTR ( <i>column expression, m[, n]</i> )                | Returns specified characters from character value starting at character position <i>m</i> , <i>n</i> characters long (If <i>m</i> is negative, the count starts from the end of the character value. If <i>n</i> is omitted, all characters to the end of the string are returned.) |

**Note:** The functions discussed in this lesson are only some of the available functions.

## Character Functions (continued)

| Function                                                                                                                                 | Purpose                                                                                                                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LENGTH ( <i>column expression</i> )                                                                                                      | Returns the number of characters in the expression                                                                                                                                                                                                                                          |
| INSTR ( <i>column expression</i> ,<br>' <i>string</i> ', [ <i>m</i> ], [ <i>n</i> ])                                                     | Returns the numeric position of a named string. Optionally, you can provide a position <i>m</i> to start searching, and the occurrence <i>n</i> of the string. <i>m</i> and <i>n</i> default to 1, meaning start the search at the beginning of the search and report the first occurrence. |
| LPAD ( <i>column expression</i> , <i>n</i> ,<br>' <i>string</i> ')<br>RPAD ( <i>column expression</i> , <i>n</i> ,<br>' <i>string</i> ') | Pads the character value right-justified to a total width of <i>n</i> character positions<br>Pads the character value left-justified to a total width of <i>n</i> character positions                                                                                                       |
| TRIM ( <i>leading trailing both</i> ,<br><i>trim character FROM</i><br><i>trim source</i> )                                              | Enables you to trim heading or trailing characters (or both) from a character string. If <i>trim character</i> or <i>trim source</i> is a character literal, you must enclose it in single quotation marks.<br>This is a feature that is available in Oracle8i and later versions.          |
| REPLACE ( <i>text</i> ,<br><i>search string</i> ,<br><i>replacement string</i> )                                                         | Searches a text expression for a character string and, if found, replaces it with a specified replacement string                                                                                                                                                                            |

S e Only

# Case-Manipulation Functions

These functions convert case for character strings:

| Function              | Result     |
|-----------------------|------------|
| LOWER('SQL Course')   | sql course |
| UPPER('SQL Course')   | SQL COURSE |
| INITCAP('SQL Course') | Sq1 Course |



Copyright © 2009, Oracle. All rights reserved.

## Case-Manipulation Functions

LOWER, UPPER, and INITCAP are the three case-conversion functions.

- **LOWER:** Converts mixed-case or uppercase character strings to lowercase
- **UPPER:** Converts mixed-case or lowercase character strings to uppercase
- **INITCAP:** Converts the first letter of each word to uppercase and remaining letters to lowercase

```
SELECT 'The job id for '||UPPER(last_name)||' is '
 ||LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM employees;
```

| EMPLOYEE DETAILS                    |
|-------------------------------------|
| 1 The job id for ABEL is sa_rep     |
| 2 The job id for DAVIES is st_clerk |
| 3 The job id for DE HAAN is ad_vp   |
| ...                                 |
| 19 The job id for WHALEN is ad_asst |
| 20 The job id for ZLOTKEY is sa_man |

# Using Case-Manipulation Functions

Display the employee number, name, and department number for employee Higgins:

```
SELECT employee_id, last_name,
 department_id FROM employees
 WHERE last_name = 'higgins';
no rows selected
```

```
SELECT employee_id, last_name, department_id
 FROM employees
 WHERE LOWER(last_name) = 'higgins';
```

|   | EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|---|-------------|-----------|---------------|
| 1 | 205         | Higgins   | 110           |

Copyright © 2009, Oracle. All rights reserved.

## Using Case-Manipulation Functions

The slide example displays the employee number, name, and department number of employee Higgins.

The WHERE clause of the first SQL statement specifies the employee name as higgins. Because all the data in the EMPLOYEES table is stored in proper case, the name higgins does not find a match in the table, and no rows are selected.

The WHERE clause of the second SQL statement specifies that the employee name in the EMPLOYEES table is compared to higgins, converting the LAST\_NAME column to lowercase for comparison purposes. Because both names are now lowercase, a match is found and one row is selected. The WHERE clause can be rewritten in the following manner to produce the same result:

```
...WHERE last_name = 'Higgins'
```

The name in the output appears as it was stored in the database. To display the name in uppercase, use the UPPER function in the SELECT statement.

```
SELECT employee_id, UPPER(last_name), department_id
 FROM employees
 WHERE INITCAP(last_name) = 'Higgins';
```

# Character-Manipulation Functions

These functions manipulate character strings:

| Function                               | Result         |
|----------------------------------------|----------------|
| CONCAT ('Hello', 'World')              | HelloWorld     |
| SUBSTR ('HelloWorld', 1, 5)            | Hello          |
| LENGTH ('HelloWorld')                  | 10             |
| INSTR ('HelloWorld', 'W')              | 6              |
| LPAD (salary, 10, '*')                 | *****24000     |
| RPAD (salary, 10, '*')                 | 24000*****     |
| REPLACE<br>('JACK and JUE', 'J', 'BL') | BLACK and BLUE |
| TRIM ('H' FROM 'HelloWorld')           | elloWorld      |



Copyright © 2009, Oracle. All rights reserved.

## Character-Manipulation Functions

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, and TRIM are the character-manipulation functions that are covered in this lesson.

- **CONCAT:** Joins values together (You are limited to using two parameters with CONCAT.)
- **SUBSTR:** Extracts a string of determined length
- **LENGTH:** Shows the length of a string as a numeric value
- **INSTR:** Finds the numeric position of a named character
- **LPAD:** Pads the character value right-justified
- **RPAD:** Pads the character value left-justified
- **TRIM:** Trims heading or trailing characters (or both) from a character string (If `trim_character` or `trim_source` is a character literal, you must enclose it in single quotation marks.)

**Note:** You can use functions such as UPPER and LOWER with ampersand substitution. For example, use `UPPER('&job_title')` so that the user does not have to enter the job title in a specific case.

# Using the Character-Manipulation Functions

```

SELECT employee_id, CONCAT(first_name, last_name) NAME, | 2
 job_id, LENGTH(last_name), |
 INSTR(last_name, 'a') "Contains 'a'?" | 3
 FROM employees
 WHERE SUBSTR(job_id, 4) = 'REP';

```

| EMPLOYEE_ID | NAME               | JOB_ID | LENGTH(LAST_NAME) | Contains 'a'? |
|-------------|--------------------|--------|-------------------|---------------|
| 1           | 202 PatFay         | MK_REP | 3                 | 2             |
| 2           | 174 EllenAbel      | SA_REP | 4                 | 0             |
| 3           | 176 JonathonTaylor | SA_REP | 6                 | 2             |
| 4           | 178 KimberlyGrant  | SA_REP | 5                 | 3             |



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the Character-Manipulation Functions

The slide example displays employee first names and last names joined together, the length of the employee last name, and the numeric position of the letter *a* in the employee last name for all employees who have the string REP contained in the job ID starting at the fourth position of the job ID.

### Example

Modify the SQL statement in the slide to display the data for those employees whose last names end with the letter *n*.

```

SELECT employee_id, CONCAT(first_name, last_name) NAME,
 LENGTH(last_name), INSTR(last_name, 'a') "Contains 'a'?"
 FROM employees
 WHERE SUBSTR(last_name, -1, 1) = 'n';

```

| EMPLOYEE_ID | NAME                 | LENGTH(LAST_NAME) | Contains 'a'? |
|-------------|----------------------|-------------------|---------------|
| 1           | 102 LexDe Haan       | 7                 | 5             |
| 2           | 200 JenniferWhalen   | 6                 | 3             |
| 3           | 201 MichaelHartstein | 9                 | 2             |

## Number Functions

- ROUND: Rounds value to specified decimal
- TRUNC: Truncates value to specified decimal
- MOD: Returns remainder of division

| Function         | Result |
|------------------|--------|
| ROUND(45.926, 2) | 45.93  |
| TRUNC(45.926, 2) | 45.92  |
| MOD(1600, 300)   | 100    |

Copyright © 2009, Oracle. All rights reserved.

### Number Functions

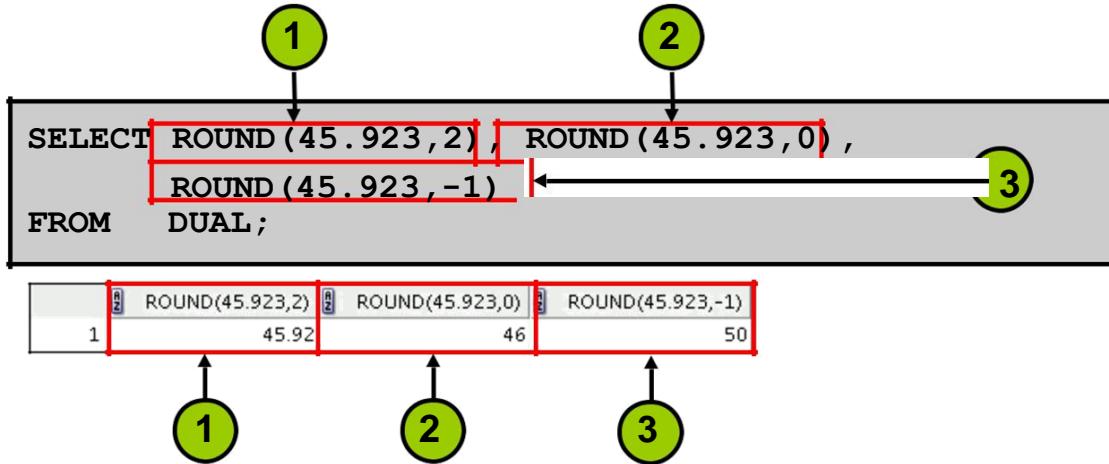
Number functions accept numeric input and return numeric values. This section describes some of the number functions.

| Function                             | Purpose                                                                                                                                                                                       |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ROUND( <i>column expression, n</i> ) | Rounds the column, expression, or value to <i>n</i> decimal places or, if <i>n</i> is omitted, no decimal places (If <i>n</i> is negative, numbers to left of the decimal point are rounded.) |
| TRUNC( <i>column expression, n</i> ) | Truncates the column, expression, or value to <i>n</i> decimal places or, if <i>n</i> is omitted, <i>n</i> defaults to zero                                                                   |
| MOD( <i>m, n</i> )                   | Returns the remainder of <i>m</i> divided by <i>n</i>                                                                                                                                         |

**Note:** This list contains only some of the available number functions.

For more information, see “Number Functions” in *Oracle SQL Reference*.

## Using the ROUND Function



**DUAL** is a dummy table that you can use to view results from functions and calculations.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ROUND Function

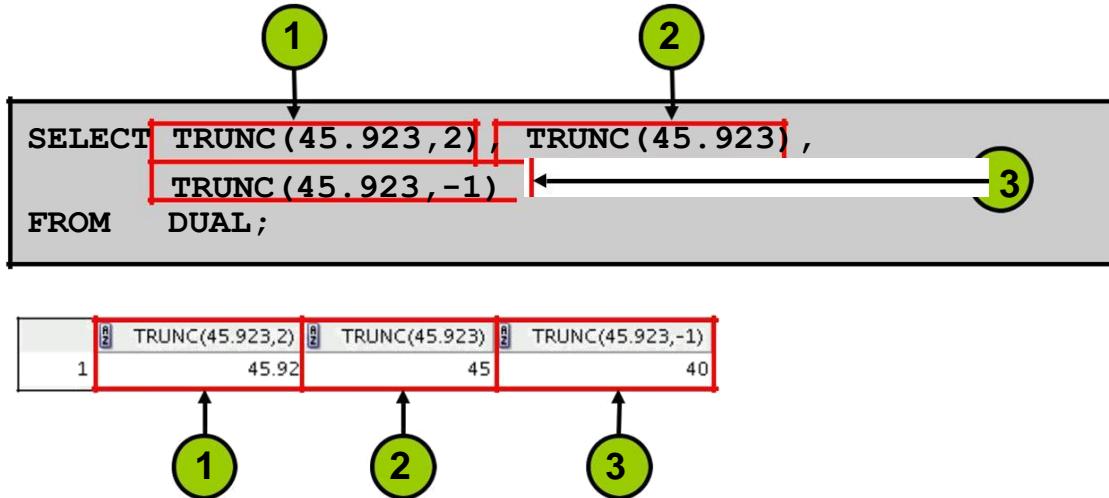
The ROUND function rounds the column, expression, or value to  $n$  decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left (rounded to the nearest unit of 10).

The ROUND function can also be used with date functions. You will see examples later in this lesson.

### DUAL Table

The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column, DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value once only (for example, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data). The DUAL table is generally used for SELECT clause syntax completeness, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from actual tables.

# Using the TRUNC Function



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## TRUNC Function

The TRUNC function truncates the column, expression, or value to  $n$  decimal places.

The TRUNC function works with arguments similar to those of the ROUND function. If the second argument is 0 or is missing, the value is truncated to zero decimal places. If the second argument is 2, the value is truncated to two decimal places. Conversely, if the second argument is -2, the value is truncated to two decimal places to the left. If the second argument is -1, the value is truncated to one decimal place to the left.

Like the ROUND function, the TRUNC function can be used with date functions.

# Using the MOD Function

For all employees with job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM employees
WHERE job_id = 'SA_REP';
```

|   | LAST_NAME | SALARY | MOD(SALARY,5000) |
|---|-----------|--------|------------------|
| 1 | Abel      | 11000  | 1000             |
| 2 | Taylor    | 8600   | 3600             |
| 3 | Grant     | 7000   | 2000             |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## MOD Function

The MOD function finds the remainder of the first argument divided by the second argument. The slide example calculates the remainder of the salary after dividing it by 5,000 for all employees whose job ID is SA\_REP.

**Note:** The MOD function is often used to determine if a value is odd or even.

# Working with Dates

- The Oracle Database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.
- The default date display format is DD-MON-RR.
  - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
  - Enables you to store 20th-century dates in the 21st century in the same way

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date < '01-FEB-88';
```

|   | LAST_NAME | HIRE_DATE |
|---|-----------|-----------|
| 1 | Whalen    | 17-SEP-87 |
| 2 | King      | 17-JUN-87 |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Oracle Date Format

The Oracle Database stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The default display and input format for any date is DD-MON-RR. Valid Oracle dates are between January 1, 4712 B.C., and December 31, 9999 A.D.

In the example in the slide, the HIRE\_DATE column output is displayed in the default format DD-MON-RR. However, dates are not stored in the database in this format. All the components of the date and time are stored. So, although a HIRE\_DATE such as 17-JUN-87 is displayed as day, month, and year, there is also *time* and *century* information associated with the date. The complete date might be June 17, 1987, 5:10:43 p.m.

## Oracle Date Format (continued)

This data is stored internally as follows:

| CENTURY | YEAR | MONTH | DAY | HOUR | MINUTE | SECOND |
|---------|------|-------|-----|------|--------|--------|
| 19      | 87   | 06    | 17  | 17   | 10     | 43     |

### Centuries and the Year 2000

When a record with a date column is inserted into a table, the *century* information is picked up from the SYSDATE function. However, when the date column is displayed on the screen, the century component is not displayed (by default).

The DATE data type always stores year information as a four-digit number internally: two digits for the century and two digits for the year. For example, the Oracle Database stores the year as 1987 or 2004, and not just as 87 or 04.

# Working with Dates

`SYSDATE` is a function that returns:

- Date
- Time



Copyright © 2009, Oracle. All rights reserved.

## **SYSDATE Function**

`SYSDATE` is a date function that returns the current database server date and time. You can use `SYSDATE` just as you would use any other column name. For example, you can display the current date by selecting `SYSDATE` from a table. It is customary to select `SYSDATE` from a dummy table called `DUAL`.

### **Example**

Display the current date using the `DUAL` table.

```
SELECT SYSDATE
FROM DUAL;
```

| SYSDATE     |
|-------------|
| 1 06-NOV-08 |

## Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.



Copyright © 2009, Oracle. All rights reserved.

### Arithmetic with Dates

Because the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates.

You can perform the following operations:

| Operation        | Result         | Description                            |
|------------------|----------------|----------------------------------------|
| date + number    | Date           | Adds a number of days to a date        |
| date – number    | Date           | Subtracts a number of days from a date |
| date – date      | Number of days | Subtracts one date from another        |
| date + number/24 | Date           | Adds a number of hours to a date       |

# Using Arithmetic Operators with Dates

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM employees
WHERE department_id = 90;
```

| LAST_NAME | WEEKS                              |
|-----------|------------------------------------|
| King      | 1116.14857473544973544973544973545 |
| Kochhar   | 998.005717592592592592592592593    |
| De Haan   | 825.14857473544973544973544973545  |



Copyright © 2009, Oracle. All rights reserved.

## Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90. It subtracts the date on which the employee was hired from the current date (SYSDATE) and divides the result by 7 to calculate the number of weeks that a worker has been employed.

**Note:** SYSDATE is a SQL function that returns the current date and time. Your results may differ from the example.

If a more current date is subtracted from an older date, the difference is a negative number.

# Date Functions

| Function       | Result                             |
|----------------|------------------------------------|
| MONTHS_BETWEEN | Number of months between two dates |
| ADD_MONTHS     | Add calendar months to date        |
| NEXT_DAY       | Next day of the date specified     |
| LAST_DAY       | Last day of the month              |
| ROUND          | Round date                         |
| TRUNC          | Truncate date                      |

Copyright © 2009, Oracle. All rights reserved.

## Date Functions

Date functions operate on Oracle dates. All date functions return a value of DATE data type except MONTHS\_BETWEEN, which returns a numeric value.

- **MONTHS\_BETWEEN(date1, date2)**: Finds the number of months between *date1* and *date2*. The result can be positive or negative. If *date1* is later than *date2*, the result is positive; if *date1* is earlier than *date2*, the result is negative. The noninteger part of the result represents a portion of the month.
- **ADD\_MONTHS(date, n)**: Adds *n* number of calendar months to *date*. The value of *n* must be an integer and can be negative.
- **NEXT\_DAY(date, 'char')**: Finds the date of the next specified day of the week ('*char*') following *date*. The value of *char* may be a number representing a day or a character string.
- **LAST\_DAY(date)**: Finds the date of the last day of the month that contains *date*.
- **ROUND(date[, 'fmt'])**: Returns *date* rounded to the unit that is specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is rounded to the nearest day.
- **TRUNC(date[, 'fmt'])**: Returns *date* with the time portion of the day truncated to the unit that is specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is truncated to the nearest day.

This list is a subset of the available date functions. The format models are covered later in this lesson. Examples of format models are month and year.

# Using Date Functions

| Function                                       | Result      |
|------------------------------------------------|-------------|
| MONTHS_BETWEEN<br>( '01-SEP-95', '11-JAN-94' ) | 19.6774194  |
| ADD_MONTHS ('11-JAN-94', 6)                    | '11-JUL-94' |
| NEXT_DAY ('01-SEP-95', 'FRIDAY')               | '08-SEP-95' |
| LAST_DAY ('01-FEB-95')                         | '28-FEB-95' |



Copyright © 2009, Oracle. All rights reserved.

## Date Functions (continued)

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for fewer than 120 months.

```
SELECT employee_id, hire_date,
 MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
 ADD_MONTHS (hire_date, 6) REVIEW,
 NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
 FROM employees
 WHERE MONTHS_BETWEEN (SYSDATE, hire_date) < 120;
```

|   | EMPLOYEE_ID | HIRE_DATE | TENURE                | REVIEW              | NEXT_DAY(HIRE_DATE,'FRIDAY') | LAST_DAY(HIRE_DATE) |
|---|-------------|-----------|-----------------------|---------------------|------------------------------|---------------------|
| 1 | 107         | 07-FEB-99 | 116.96930966248506... | 07-AUG-99 12-FEB-99 | 28-FEB-99                    |                     |
| 2 | 124         | 16-NOV-99 | 107.67898708183990... | 16-MAY-00 19-NOV-99 | 30-NOV-99                    |                     |
| 3 | 149         | 29-JAN-00 | 105.25963224313022... | 29-JUL-00 04-FEB-00 | 31-JAN-00                    |                     |
| 4 | 178         | 24-MAY-99 | 113.42092256571087... | 24-NOV-99 28-MAY-99 | 31-MAY-99                    |                     |

# Using Date Functions

Assume SYSDATE = '25-JUL-03':

| Function                | Result    |
|-------------------------|-----------|
| ROUND(SYSDATE, 'MONTH') | 01-AUG-03 |
| ROUND(SYSDATE, 'YEAR')  | 01-JAN-04 |
| TRUNC(SYSDATE, 'MONTH') | 01-JUL-03 |
| TRUNC(SYSDATE, 'YEAR')  | 01-JAN-03 |



Copyright © 2009, Oracle. All rights reserved.

## Date Functions (continued)

The ROUND and TRUNC functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month.

### Example

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and start month using the ROUND and TRUNC functions.

```
SELECT employee_id, hire_date,
 ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
 FROM employees
 WHERE hire_date LIKE '%97';
```

| EMPLOYEE_ID | HIRE_DATE     | ROUND(HIRE_DATE,'MONTH') | TRUNC(HIRE_DATE,'MONTH') |
|-------------|---------------|--------------------------|--------------------------|
| 1           | 202 17-AUG-97 | 01-SEP-97                | 01-AUG-97                |
| 2           | 142 29-JAN-97 | 01-FEB-97                | 01-JAN-97                |

## Practice 3: Overview of Part 1

This practice covers the following topics:

- Writing a query that displays the current date
- Creating queries that require the use of the numeric, character, and date functions
- Performing calculations of years and months of service for an employee



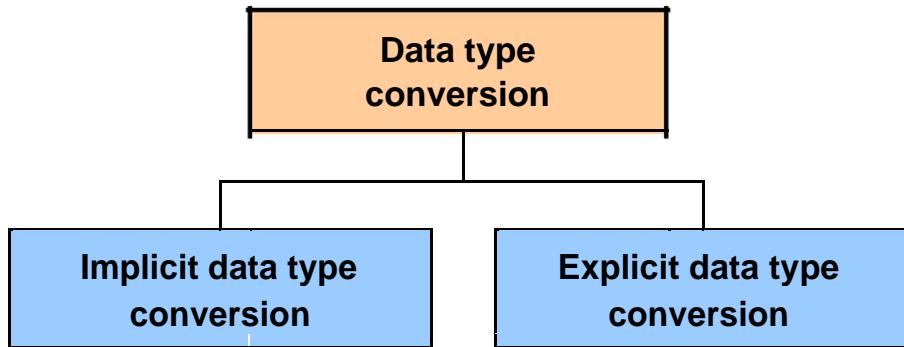
Copyright © 2009, Oracle. All rights reserved.

### Practice 3: Overview of Part 1

Part 1 of this lesson's practice provides a variety of exercises that use the different functions that are available for the character, number, and date data types.

For Part 1, complete questions 1–6 at the end of this lesson.

# Conversion Functions



Copyright © 2009, Oracle. All rights reserved.

## Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle Database can be defined using ANSI, DB2, and SQL/DS data types. However, the Oracle server internally converts such data types to Oracle data types.

In some cases, the Oracle server uses data of one data type where it expects data of a different data type. When this happens, the Oracle server can automatically convert the data to the expected data type. This data type conversion can be done *implicitly* by the Oracle server or *explicitly* by the user. Implicit data type conversions work according to the rules that are explained in the next two slides.

Explicit data type conversions are done by using the conversion functions. Conversion functions convert a value from one data type to another. Generally, the form of the function names follows the convention *data type TO data type*. The first data type is the input data type; the second data type is the output.

**Note:** Although implicit data type conversion is available, it is recommended that you do explicit data type conversion to ensure the reliability of your SQL statements.

## Implicit Data Type Conversion

For assignments, the Oracle server can automatically convert the following:

| From             | To       |
|------------------|----------|
| VARCHAR2 or CHAR | NUMBER   |
| VARCHAR2 or CHAR | DATE     |
| NUMBER           | VARCHAR2 |
| DATE             | VARCHAR2 |



Copyright © 2009, Oracle. All rights reserved.

### Implicit Data Type Conversion

The assignment succeeds if the Oracle server can convert the data type of the value used in the assignment to that of the assignment target.

For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string `'01-JAN-90'` to a date.

## Implicit Data Type Conversion

For expression evaluation, the Oracle Server can automatically convert the following:

| From             | To     |
|------------------|--------|
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE   |



Copyright © 2009, Oracle. All rights reserved.

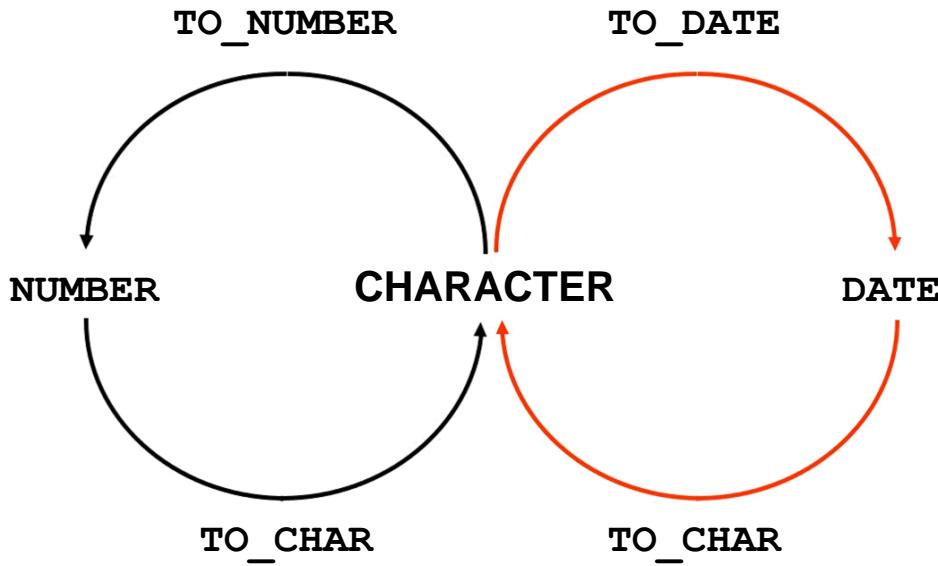
### Implicit Data Type Conversion (continued)

In general, the Oracle server uses the rule for expressions when a data type conversion is needed in places that are not covered by a rule for assignment conversions.

For example, the expression `salary = '20000'` results in the implicit conversion of the string '`20000`' to the number `20000`.

**Note:** CHAR to NUMBER conversions succeed only if the character string represents a valid number.

# Explicit Data Type Conversion



ORACLE

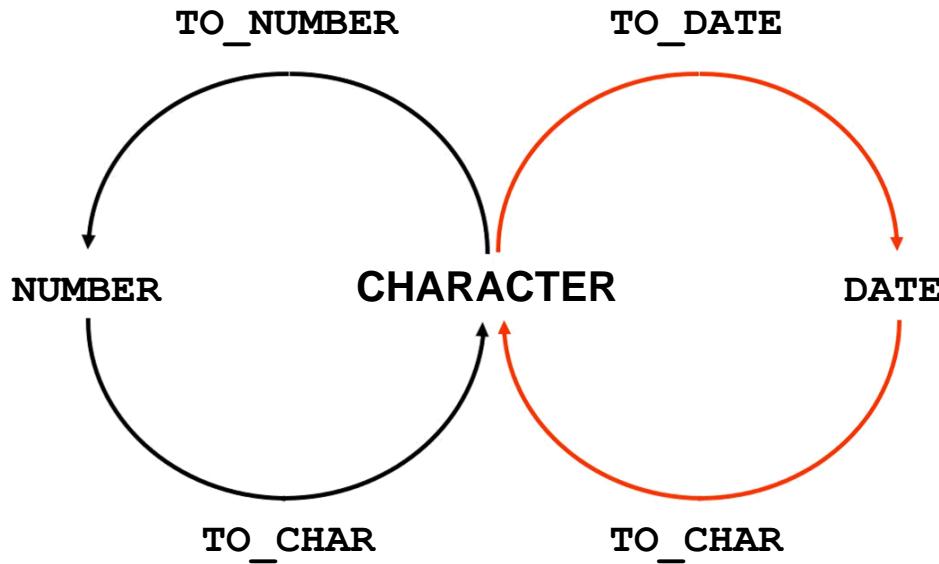
Copyright © 2009, Oracle. All rights reserved.

## Explicit Data Type Conversion

SQL provides three functions to convert a value from one data type to another:

| Function                                                | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>TO_CHAR(number date, [ fmt ], [nlsparams])</code> | <p>Converts a number or date value to a VARCHAR2 character string with format model <i>fmt</i>.</p> <p><b>Number conversion:</b> The <i>nlsparams</i> parameter specifies the following characters, which are returned by number format elements:</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Decimal character</li> <li><input type="checkbox"/> Group separator</li> <li><input type="checkbox"/> Local currency symbol</li> <li><input type="checkbox"/> International currency symbol</li> </ul> <p>If <i>nlsparams</i> or any other parameter is omitted, this function uses the default parameter values for the session.</p> |

# Explicit Data Type Conversion



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Explicit Data Type Conversion (continued)

| Function                                                | Purpose                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>TO_CHAR(number date, [ fmt ], [nlsparams])</code> | <b>Date conversion:</b> The <code>nlsparams</code> parameter specifies the language in which month and day names and abbreviations are returned. If this parameter is omitted, this function uses the default date languages for the session.                                                                         |
| <code>TO_NUMBER(char, [ fmt ], [nlsparams])</code>      | Converts a character string containing digits to a number in the format specified by the optional format model <code>fmt</code> . The <code>nlsparams</code> parameter has the same purpose in this function as in the <code>TO_CHAR</code> function for number conversion.                                           |
| <code>TO_DATE(char, [ fmt ], [nlsparams])</code>        | Converts a character string representing a date to a date value according to the <code>fmt</code> that is specified. If <code>fmt</code> is omitted, the format is DD-MON-YY. The <code>nlsparams</code> parameter has the same purpose in this function as in the <code>TO_CHAR</code> function for date conversion. |

## Explicit Data Type Conversion (continued)

**Note:** The list of functions mentioned in this lesson includes only some of the available conversion functions.

For more information, see “Conversion Functions” in *Oracle SQL Reference*.

# Using the TO\_CHAR Function with Dates

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed by single quotation marks
- Is case sensitive
- Can include any valid date format element
- Has an `fm` element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

Copyright © 2009, Oracle. All rights reserved.

## Displaying a Date in a Specific Format

Previously, all Oracle date values were displayed in the DD-MON-YY format. You can use the `TO_CHAR` function to convert a date from this default format to one that you specify.

### Guidelines

- The format model must be enclosed by single quotation marks and is case sensitive.
- The format model can include any valid date format element. Be sure to separate the date value from the format model by a comma.
- The names of days and months in the output are automatically padded with blanks.
- To remove padded blanks or to suppress leading zeros, use the fill mode `fm` element.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees
WHERE last_name = 'Higgins';
```

| EMPLOYEE_ID | MONTH_HIRED |
|-------------|-------------|
| 1           | 205 06/94   |

## Elements of the Date Format Model

| Element | Result                                           |
|---------|--------------------------------------------------|
| YYYY    | Full year in numbers                             |
| YEAR    | Year spelled out (in English)                    |
| MM      | Two-digit value for month                        |
| MONTH   | Full name of the month                           |
| MON     | Three-letter abbreviation of the month           |
| DY      | Three-letter abbreviation of the day of the week |
| DAY     | Full name of the day of the week                 |
| DD      | Numeric day of the month                         |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Sample Format Elements of Valid Date Formats

| Element                      | Description                                                                                                           |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| SCC or CC                    | Century; server prefixes B.C. date with -                                                                             |
| Years in dates YYYY or SYYYY | Year; server prefixes B.C. date with -                                                                                |
| YYY or YY or Y               | Last three, two, or one digits of year                                                                                |
| Y,YYY                        | Year with comma in this position                                                                                      |
| IYYY, IYY, IY, I             | Four-, three-, two-, or one-digit year based on the ISO standard                                                      |
| SYEAR or YEAR                | Year spelled out; server prefixes B.C. date with -                                                                    |
| BC or AD                     | Indicates B.C. or A.D. year                                                                                           |
| B.C. or A.D.                 | Indicates B.C. or A.D. year using periods                                                                             |
| Q                            | Quarter of year                                                                                                       |
| MM                           | Month: two-digit value                                                                                                |
| MONTH                        | Name of month padded with blanks to length of nine characters                                                         |
| MON                          | Name of month, three-letter abbreviation                                                                              |
| RM                           | Roman numeral month                                                                                                   |
| WW or W                      | Week of year or month                                                                                                 |
| DDD or DD or D               | Day of year, month, or week                                                                                           |
| DAY                          | <small>Day of day; three-letter abbreviation</small><br>Name of day padded with blanks to a length of nine characters |
| J                            | Julian day; the number of days since December 31, 4713 B.C.                                                           |

## Elements of the Date Format Model

- Time elements format the time portion of the date:

|                            |                          |
|----------------------------|--------------------------|
| <code>HH24:MI:SS AM</code> | <code>15:45:32 PM</code> |
|----------------------------|--------------------------|

- Add character strings by enclosing them in double quotation marks:

|                            |                            |
|----------------------------|----------------------------|
| <code>DD "of" MONTH</code> | <code>12 of OCTOBER</code> |
|----------------------------|----------------------------|

- Number suffixes spell out numbers:

|                     |                         |
|---------------------|-------------------------|
| <code>ddspth</code> | <code>fourteenth</code> |
|---------------------|-------------------------|

**ORACLE®**

Copyright © 2009, Oracle. All rights reserved.

### Date Format Elements: Time Formats

Use the formats that are listed in the following tables to display time information and literals and to change numerals to spelled numbers.

| Element            | Description                                 |
|--------------------|---------------------------------------------|
| AM or PM           | Meridian indicator                          |
| A.M. or P.M.       | Meridian indicator with periods             |
| HH or HH12 or HH24 | Hour of day, or hour (1–12), or hour (0–23) |
| MI                 | Minute (0–59)                               |
| SS                 | Second (0–59)                               |
| SSSS               | Seconds past midnight (0–86399)             |

## Other Formats

| Element  | Description                                |
|----------|--------------------------------------------|
| / . ,    | Punctuation is reproduced in the result.   |
| “of the” | Quoted string is reproduced in the result. |

## Specifying Suffixes to Influence Number Display

| Element      | Description                                                  |
|--------------|--------------------------------------------------------------|
| TH           | Ordinal number (for example, DDTH for 4TH)                   |
| SP           | Spelled-out number (for example, DDS for FOUR)               |
| SPTH or THSP | Spelled-out ordinal numbers (for example, DDSPTH for FOURTH) |

## Using the TO\_CHAR Function with Dates

```
SELECT last_name,
 TO_CHAR(hire_date, 'fmDD Month YYYY')
 AS HIREDATE
FROM employees;
```

| LAST_NAME   | HIREDATE          |
|-------------|-------------------|
| 1 Whalen    | 17 September 1987 |
| 2 Hartstein | 17 February 1996  |
| 3 Fay       | 17 August 1997    |
| 4 Higgins   | 7 June 1994       |
| 5 Gietz     | 7 June 1994       |
| ...         |                   |
| 19 Taylor   | 24 March 1998     |
| 20 Grant    | 24 May 1999       |



Copyright © 2009, Oracle. All rights reserved.

## Using the TO\_CHAR Function with Dates

The SQL statement in the slide displays the last names and hire dates for all the employees. The hire date appears as 17 June 1987.

### Example

Modify the slide example to display the dates in a format that appears as “Seventeenth of June 1987 12:00:00 AM.”

```
SELECT last_name,
 TO_CHAR(hire_date,
 'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
 AS HIREDATE
FROM employees;
```

| LAST_NAME   | HIREDATE                                  |
|-------------|-------------------------------------------|
| 1 Whalen    | Seventeenth of September 1987 12:00:00 AM |
| 2 Hartstein | Seventeenth of February 1996 12:00:00 AM  |
| ...         |                                           |
| 20 Grant    | Twenty-Fourth of May 1999 12:00:00 AM     |

Notice that the month follows the format model specified; in other words, the first letter is capitalized and the rest are lowercase.

## Using the TO\_CHAR Function with Numbers

```
TO_CHAR(number, 'format model')
```

These are some of the format elements that you can use with the TO\_CHAR function to display a number value as a character:

| Element | Result                                  |
|---------|-----------------------------------------|
| 9       | Represents a number                     |
| 0       | Forces a zero to be displayed           |
| \$      | Places a floating dollar sign           |
| L       | Uses the floating local currency symbol |
| .       | Prints a decimal point                  |
| ,       | Prints a comma as thousands indicator   |



Copyright © 2009, Oracle. All rights reserved.

## Using the TO\_CHAR Function with Numbers

When working with number values such as character strings, you should convert those numbers to the character data type using the TO\_CHAR function, which translates a value of NUMBER data type to VARCHAR2 data type. This technique is especially useful with concatenation.

## Using the TO\_CHAR Function with Numbers (continued)

### Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

| Element | Description                                                                                                                | Example             | Result             |
|---------|----------------------------------------------------------------------------------------------------------------------------|---------------------|--------------------|
| 9       | Numeric position (number of 9s determine display width)                                                                    | 999999              | 1234               |
| 0       | Display leading zeros                                                                                                      | 099999              | 001234             |
| \$      | Floating dollar sign                                                                                                       | \$999999            | \$1234             |
| L       | Floating local currency symbol                                                                                             | L999999             | FF1234             |
| D       | Returns in the specified position the decimal character. The default is a period (.).                                      | 99D99               | 99.99              |
| .       | Decimal point in position specified                                                                                        | 999999.99           | 1234.00            |
| G       | Returns the group separator in the specified position. You can specify multiple group separators in a number format model. | 9,999               | 9G999              |
| ,       | Comma in position specified                                                                                                | 999,999             | 1,234              |
| MI      | Minus signs to right (negative values)                                                                                     | 999999MI            | 1234-              |
| PR      | Parenthesize negative numbers                                                                                              | 999999PR            | <1234>             |
| EEEE    | Returns in the specified position the "Euro" (or Scientific notation (format must specify four Es))                        | U9999<br>99.999EEEE | €1234<br>1.234E+03 |
|         | other) dual currency                                                                                                       |                     |                    |
| V       | Multiply by 10 $n$ times ( $n$ = number of 9s after V)                                                                     | 9999V99             | 123400             |
| S       | Returns the negative or positive value                                                                                     | S9999               | -1234 or           |
| B       | Display zero values as blank, not 0                                                                                        | B9999.99            | 1234.00            |

## Using the TO\_CHAR Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM employees
WHERE last_name = 'Ernst';
```

|   | SALARY     |
|---|------------|
| 1 | \$6,000.00 |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### Guidelines

- The Oracle server displays a string of number signs (#) in place of a whole number whose digits exceed the number of digits that is provided in the format model.
- The Oracle server rounds the stored decimal value to the number of decimal places that is provided in the format model.

## Using the TO\_NUMBER and TO\_DATE Functions

- Convert a character string to a number format using the TO\_NUMBER function:

```
TO_NUMBER(char[, 'format_model'])
```

- Convert a character string to a date format using the TO\_DATE function:

```
TO_DATE(char[, 'format_model'])
```

- These functions have an *fx* modifier. This modifier specifies the exact matching for the character argument and date format model of a TO\_DATE function.



Copyright © 2009, Oracle. All rights reserved.

## Using the TO\_NUMBER and TO\_DATE Functions

You may want to convert a character string to either a number or a date. To accomplish this task, use the TO\_NUMBER or TO\_DATE functions. The format model that you choose is based on the previously demonstrated format elements.

The *fx* modifier specifies exact matching for the character argument and date format model of a TO\_DATE function:

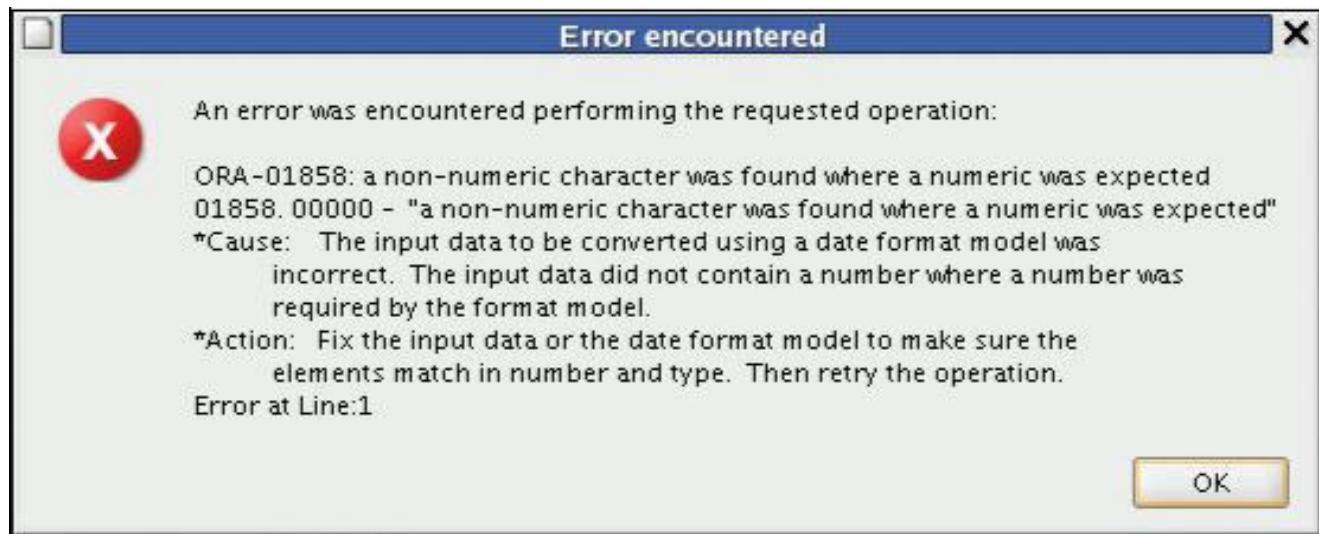
- Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.
- The character argument cannot have extra blanks. Without *fx*, Oracle ignores extra blanks.
- Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without *fx*, numbers in the character argument can omit leading zeros.

## Using the TO\_NUMBER and TO\_DATE Functions (continued)

### Example

Display the name and hire date for all employees who started on May 24, 1999. There are two spaces after the month *May* and the number *24* in the following example. Because the *fx* modifier is used, an exact match is required and the spaces after the word *May* are not recognized:

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```



To fix the above error, change *fx* to *fm* in the query.

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fmMonth DD, YYYY');
```

| LAST_NAME | HIRE_DATE |
|-----------|-----------|
| Grant     | 24-MAY-99 |

## RR Date Format

| Current Year | Specified Date | RR Format | YY Format |
|--------------|----------------|-----------|-----------|
| 1995         | 27-OCT-95      | 1995      | 1995      |
| 1995         | 27-OCT-17      | 2017      | 1917      |
| 2001         | 27-OCT-17      | 2017      | 2017      |
| 2001         | 27-OCT-95      | 1995      | 2095      |

|                                        |       | If the specified two-digit year is:                     |                                                          |
|----------------------------------------|-------|---------------------------------------------------------|----------------------------------------------------------|
|                                        |       | 0–49                                                    | 50–99                                                    |
| If two digits of the current year are: | 0–49  | The return date is in the current century               | The return date is in the century before the current one |
|                                        | 50–99 | The return date is in the century after the current one | The return date is in the current century                |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### RR Date Format Element

The RR date format is similar to the YY element, but you can use it to specify different centuries. Use the RR date format element instead of YY so that the century of the return value varies according to the specified two-digit year and the last two digits of the current year. The table in the slide summarizes the behavior of the RR element.

| Current Year | Given Date | Interpreted (RR) | Interpreted (YY) |
|--------------|------------|------------------|------------------|
| 1994         | 27-OCT-95  | 1995             | 1995             |
| 1994         | 27-OCT-17  | 2017             | 1917             |
| 2001         | 27-OCT-17  | 2017             | 2017             |

## RR Date Format: Example

To find employees hired before 1990, use the RR date format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

|   | LAST_NAME | TO_CHAR(HIRE_DATE,'DD-MON-YYYY') |
|---|-----------|----------------------------------|
| 1 | Whalen    | 17-Sep-1987                      |
| 2 | King      | 17-Jun-1987                      |
| 3 | Kochhar   | 21-Sep-1989                      |



Copyright © 2009, Oracle. All rights reserved.

## RR Date Format: Example

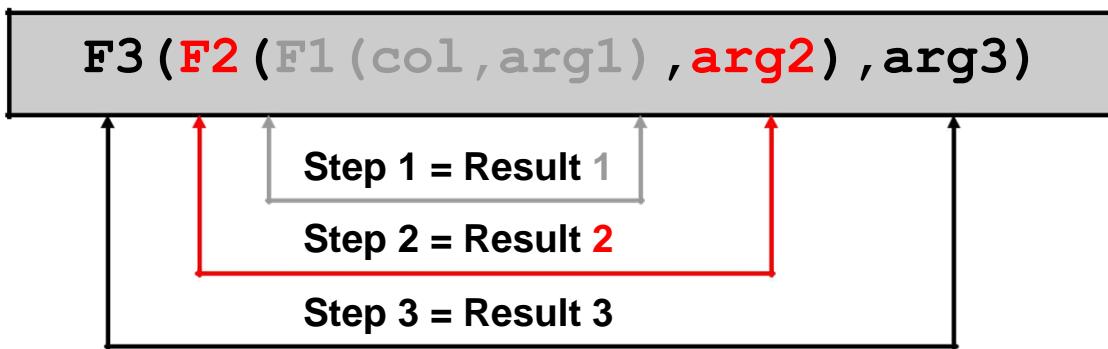
To find employees who were hired before 1990, the RR format can be used. Because the current year is greater than 1999, the RR format interprets the year portion of the date from 1950 to 1999.

The following command, on the other hand, results in no rows being selected because the YY format interprets the year portion of the date in the current century (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM employees
WHERE TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

# Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from the deepest level to the least deep level.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Nesting Functions

Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.

# Nesting Functions

```
SELECT last_name,
 UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
 FROM employees
 WHERE department_id = 60;
```

|   | LAST_NAME | UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US')) |
|---|-----------|--------------------------------------------|
| 1 | Hunold    | HUNOLD_US                                  |
| 2 | Ernst     | ERNST_US                                   |
| 3 | Lorentz   | LORENTZ_US                                 |



Copyright © 2009, Oracle. All rights reserved.

## Nesting Functions (continued)

The slide example displays the last names of employees in department 60. The evaluation of the SQL statement involves three steps:

1. The inner function retrieves the first eight characters of the last name.

Result1 = SUBSTR (LAST\_NAME, 1, 8)

2. The outer function concatenates the result with \_US.

Result2 = CONCAT(Result1, '\_US')

3. The outermost function converts the results to uppercase.

The entire expression becomes the column heading because no column alias was given.

## Example

Display the date of the next Friday that is six months from the hire date. The resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS
 (hire_date, 6), 'FRIDAY'),
 'fmDay, Month DDth, YYYY')
 "Next 6 Month Review"
 FROM employees
 ORDER BY hire_date;
```

## General Functions

The following functions work with any data type and pertain to using nulls:

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)
- NULLIF (expr1, expr2)
- COALESCE (expr1, expr2, ..., exprn)



Copyright © 2009, Oracle. All rights reserved.

### General Functions

These functions work with any data type and pertain to the use of null values in the expression list.

| Function | Description                                                                                                                |
|----------|----------------------------------------------------------------------------------------------------------------------------|
| NVL      | Converts a null value to an actual value                                                                                   |
| NVL2     | If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type. |
| NULLIF   | Compares two expressions and returns null if they are equal; returns the first expression if they are not equal            |
| COALESCE | Returns the first non-null expression in the expression list                                                               |

**Note:** For more information about the hundreds of functions available, see “Functions” in *Oracle SQL Reference*.

# NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match:
  - NVL(*commission\_pct*, 0)
  - NVL(*hire\_date*, '01-JAN-97')
  - NVL(*job\_id*, 'No Job Yet')



Copyright © 2009, Oracle. All rights reserved.

## NVL Function

To convert a null value to an actual value, use the NVL function.

### Syntax

`NVL (expr1, expr2)`

In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the target value for converting the null

You can use the NVL function to convert any data type, but the return value is always the same as the data type of *expr1*.

### NVL Conversions for Various Data Types

| Data Type        | Conversion Example                                       |
|------------------|----------------------------------------------------------|
| NUMBER           | <code>NVL(<i>number_column</i>, 9)</code>                |
| DATE             | <code>NVL(<i>date_column</i>, '01-JAN-95')</code>        |
| CHAR or VARCHAR2 | <code>NVL(<i>character_column</i>, 'Unavailable')</code> |

# Using the NVL Function

```
SELECT last_name, salary, NVL(commission_pct, 0)
 (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

| LAST_NAME | SALARY | NVL(COMMISSION_PCT,0) | AN_SAL |
|-----------|--------|-----------------------|--------|
| Whalen    | 4400   | 0                     | 52800  |
| Hartstein | 13000  | 0                     | 156000 |
| Fay       | 6000   | 0                     | 72000  |
| Higgins   | 12000  | 0                     | 144000 |
| Gietz     | 8300   | 0                     | 99600  |
| King      | 24000  | 0                     | 288000 |
| Kochhar   | 17000  | 0                     | 204000 |
| De Haan   | 17000  | 0                     | 204000 |
| ...       |        |                       |        |



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the NVL Function

To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to the result:

```
SELECT last_name, salary, commission_pct,
 (salary*12) + (salary*12*commission_pct) AN_SAL
FROM employees;
```

| LAST_NAME | SALARY | COMMISSION_PCT | AN_SAL |
|-----------|--------|----------------|--------|
| Whalen    | 4400   | (null)         | (null) |
| ...       |        |                |        |
| Zlotkey   | 10500  | 0.2            | 151200 |
| Abel      | 11000  | 0.3            | 171600 |
| Taylor    | 8600   | 0.2            | 123840 |
| Grant     | 7000   | 0.15           | 96600  |

Notice that the annual compensation is calculated for only those employees who earn a commission. If any column value in an expression is null, the result is null. To calculate values for all employees, you must convert the null value to a number before applying the arithmetic operator. In the example in the slide, the NVL function is used to convert null values to zero.

# Using the NVL2 Function

```

SELECT last_name, salary, commission_pct, 1
 NVL2(commission_pct,
 'SAL+COMM', 'SAL') income 2
 FROM employees WHERE department_id IN (50, 80);

```

| LAST_NAME | SALARY | COMMISSION_PCT | INCOME   |
|-----------|--------|----------------|----------|
| Mourgos   | 5800   | (null)         | SAL      |
| Rajs      | 3500   | (null)         | SAL      |
| Davies    | 3100   | (null)         | SAL      |
| Matos     | 2600   | (null)         | SAL      |
| Vargas    | 2500   | (null)         | SAL      |
| Zlotkey   | 10500  | 0.2            | SAL+COMM |
| Abel      | 11000  | 0.3            | SAL+COMM |
| Taylor    | 8600   | 0.2            | SAL+COMM |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the NVL2 Function

The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned.

### Syntax

```
NVL2(expr1, expr2, expr3)
```

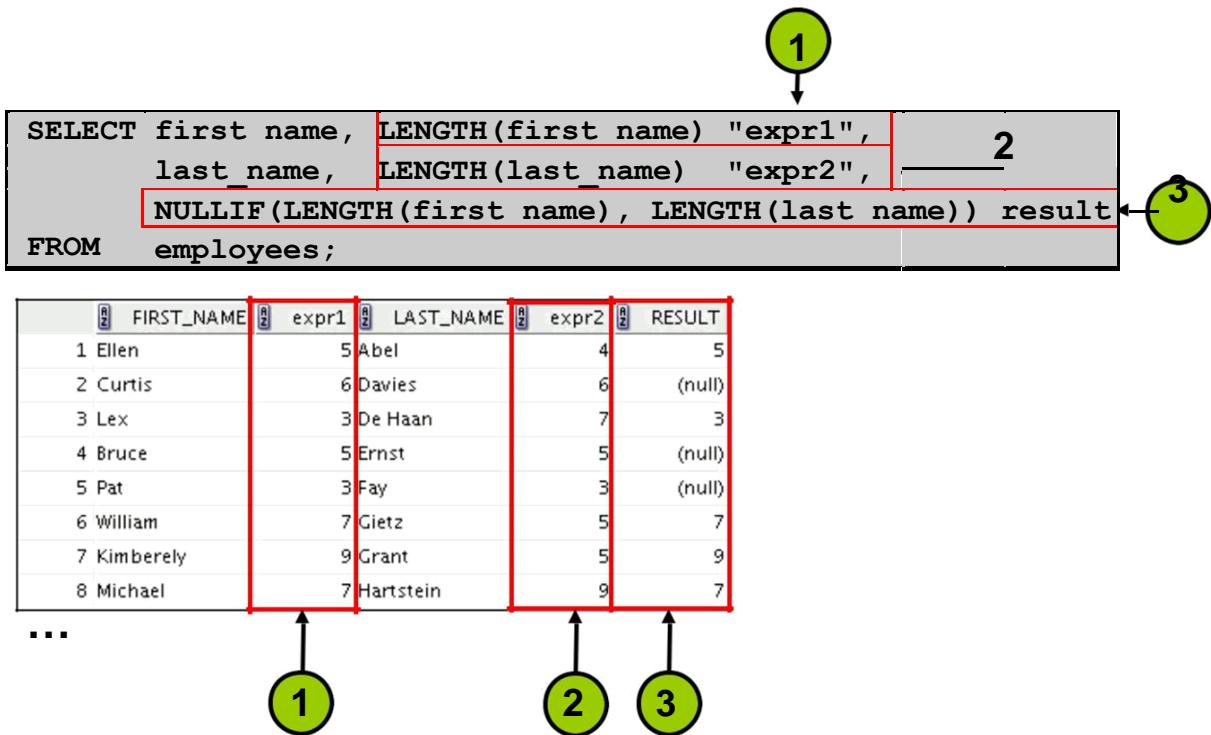
In the syntax:

- `expr1` is the source value or expression that may contain null
- `expr2` is the value that is returned if `expr1` is not null
- `expr3` is the value that is returned if `expr1` is null

In the example shown in the slide, the `COMMISSION_PCT` column is examined. If a value is detected, the second expression of `SAL+COMM` is returned. If the `COMMISSION_PCT` column holds a null value, the third expression of `SAL` is returned.

The argument `expr1` can have any data type. The arguments `expr2` and `expr3` can have any data types except `LONG`. If the data types of `expr2` and `expr3` are different, the Oracle server converts `expr3` to the data type of `expr2` before comparing them unless `expr3` is a null constant. In the latter case, a data type conversion is not necessary. The data type of the return value is always the same as the data type of `expr2`, unless `expr2` is character data, in which case the return value's data type is `VARCHAR2`.

# Using the NULLIF Function



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the NULLIF Function

The NULLIF function compares two expressions. If they are equal, the function returns null. If they are not equal, the function returns the first expression. You cannot specify the literal NULL for the first expression.

### Syntax

`NULLIF (expr1, expr2)`

In the syntax:

- `expr1` is the source value compared to `expr2`
- `expr2` is the source value compared with `expr1` (If it is not equal to `expr1`, `expr1` is returned.)

In the example shown in the slide, the length of the first name in the EMPLOYEES table is compared to the length of the last name in the EMPLOYEES table. When the lengths of the names are equal, a null value is displayed. When the lengths of the names are not equal, the length of the first name is displayed.

**Note:** The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed on a subsequent page:

`CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END`

## Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternate values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.



Copyright © 2009, Oracle. All rights reserved.

### Using the COALESCE Function

The COALESCE function returns the first non-null expression in the list.

#### Syntax

```
COALESCE (expr1, expr2, ... exprn)
```

In the syntax:

- *expr1* returns this expression if it is not null
- *expr2* returns this expression if the first expression is null and this expression is not null
- *exprn* returns this expression if the preceding expressions are null

All expressions must be of the same data type.

## Using the COALESCE Function

```
SELECT last_name,
 COALESCE(manager_id,commission_pct, -1) comm
FROM employees
ORDER BY commission_pct;
```

| LAST_NAME | COMM |
|-----------|------|
| Grant     | 149  |
| Taylor    | 149  |
| Zlotkey   | 100  |
| Abel      | 149  |
| King      | -1   |
| Kochhar   | 100  |
| De Haan   | 100  |
| Hunold    | 102  |
| ...       |      |



Copyright © 2009, Oracle. All rights reserved.

### Using the COALESCE Function (continued)

In the example shown in the slide, if the MANAGER\_ID value is not null, it is displayed. If the MANAGER\_ID value is null, then the COMMISSION\_PCT is displayed. If the MANAGER\_ID and COMMISSION\_PCT values are null, then the value -1 is displayed.

# Conditional Expressions

- Provide the use of IF-THEN-ELSE logic within a SQL statement
- Use two methods:
  - CASE expression
  - DECODE function



Copyright © 2009, Oracle. All rights reserved.

## Conditional Expressions

Two methods used to implement conditional processing (IF-THEN-ELSE logic) in a SQL statement are the CASE expression and the DECODE function.

**Note:** The CASE expression complies with ANSI SQL. The DECODE function is specific to Oracle syntax.

## CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
 [WHEN comparison_expr2 THEN return_expr2
 WHEN comparison_exprn THEN return_exprn
 ELSE else_expr]
END
```



Copyright © 2009, Oracle. All rights reserved.

### CASE Expression

CASE expressions let you use IF-THEN-ELSE logic in SQL statements without having to invoke procedures.

In a simple CASE expression, the Oracle server searches for the first WHEN ... THEN pair for which expr is equal to comparison\_expr and returns return\_expr. If none of the WHEN ... THEN pairs meet this condition, and if an ELSE clause exists, then the Oracle server returns else\_expr. Otherwise, the Oracle server returns null. You cannot specify the literal NULL for all the return\_exprs and the else\_expr.

All of the expressions (expr, comparison\_expr, and return\_expr) must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2.

# Using the CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
SELECT last_name, job_id, salary,
 CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
 WHEN 'ST_CLERK' THEN 1.15*salary
 WHEN 'SA REP' THEN 1.20*salary
 ELSE salary END "REVISED_SALARY"
FROM employees;
```

|    | LAST_NAME | JOB_ID   | SALARY | REVISED_SALARY |
|----|-----------|----------|--------|----------------|
| 11 | Lorentz   | IT_PROG  | 4200   | 4620           |
| 12 | Mourgos   | ST_MAN   | 5800   | 5800           |
| 13 | Rajs      | ST_CLERK | 3500   | 4025           |
| 14 | Davies    | ST_CLERK | 3100   | 3565           |
| 15 | Matos     | ST_CLERK | 2600   | 2990           |
| 16 | Vargas    | ST_CLERK | 2500   | 2875           |
| 17 | Zlotkey   | SA_MAN   | 10500  | 10500          |
| 18 | Abel      | SA REP   | 11000  | 13200          |
| 19 | Taylor    | SA REP   | 8600   | 10320          |
| 20 | Grant     | SA REP   | 7000   | 8400           |

Copyright © 2009, Oracle. All rights reserved.

## Using the CASE Expression

In the SQL statement in the slide, the value of JOB\_ID is decoded. If JOB\_ID is IT\_PROG, the salary increase is 10%; if JOB\_ID is ST\_CLERK, the salary increase is 15%; if JOB\_ID is SA REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be written with the DECODE function.

The following is an example of a searched CASE expression. In a searched CASE expression, the search occurs from left to right until an occurrence of the listed condition is found, and then it returns the return expression. If no condition is found to be true, and if an ELSE clause exists, the return expression in the ELSE clause is returned; otherwise, NULL is returned.

```
SELECT last_name, salary,
 (CASE WHEN salary<5000 THEN 'Low'
 WHEN salary<10000 THEN 'Medium'
 WHEN salary<20000 THEN 'Good'
 ELSE 'Excellent'
 END) qualified_salary
FROM employees;
```

## DECODE Function

Facilitates conditional inquiries by doing the work of a CASE expression or an IF-THEN-ELSE statement:

```
DECODE(col|expression, search1, result1
 [, search2, result2, ...]
 [, default])
```



Copyright © 2009, Oracle. All rights reserved.

### DECODE Function

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic that is used in various languages. The DECODE function decodes *expression* after comparing it to each *search* value. If the expression is the same as *search*, *result* is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

# Using the DECODE Function

```
SELECT last_name, job_id, salary,
 DECODE(job_id, 'IT_PROG', 1.10*salary,
 'ST_CLERK', 1.15*salary,
 'SA REP', 1.20*salary,
 salary)
 REVISED_SALARY
FROM employees;
```

|    | LAST_NAME | JOB_ID   | SALARY | REVISED_SALARY |
|----|-----------|----------|--------|----------------|
| 11 | Lorentz   | IT_PROG  | 4200   | 4620           |
| 12 | Mourgos   | ST_MAN   | 5800   | 5800           |
| 13 | Rajs      | ST_CLERK | 3500   | 4025           |
| 14 | Davies    | ST_CLERK | 3100   | 3565           |
| 15 | Matos     | ST_CLERK | 2600   | 2990           |
| 16 | Vargas    | ST_CLERK | 2500   | 2875           |
| 17 | Zlotkey   | SA_MAN   | 10500  | 10500          |
| 18 | Abel      | SA REP   | 11000  | 13200          |
| 19 | Taylor    | SA REP   | 8600   | 10320          |
| 20 | Grant     | SA REP   | 7000   | 8400           |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the DECODE Function

In the SQL statement in the slide, the value of JOB\_ID is tested. If JOB\_ID is IT\_PROG, the salary increase is 10%; if JOB\_ID is ST\_CLERK, the salary increase is 15%; if JOB\_ID is SA REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be expressed in pseudocode as an IF-THEN-ELSE statement:

```
IF job_id = 'IT_PROG' THEN salary = salary*1.10
IF job_id = 'ST_CLERK' THEN salary = salary*1.15
IF job_id = 'SA REP' THEN salary = salary*1.20
ELSE salary = salary
```

# Using the DECODE Function

Display the applicable tax rate for each employee in department 80:

```
SELECT last_name, salary,
 DECODE (TRUNC(salary/2000, 0),
 0, 0.00,
 1, 0.09,
 2, 0.20,
 3, 0.30,
 4, 0.40,
 5, 0.42,
 6, 0.44,
 0.45) TAX_RATE
 FROM employees
 WHERE department_id = 80;
```

Copyright © 2009, Oracle. All rights reserved.

## Using the DECODE function (continued)

This slide shows another example using the DECODE function. In this example, you determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as follows:

| <i>Monthly Salary Range</i> | <i>Tax Rate</i> |
|-----------------------------|-----------------|
| \$0.00–1,999.99             | 00%             |
| \$2,000.00–3,999.99         | 09%             |
| \$4,000.00–5,999.99         | 20%             |
| \$6,000.00–7,999.99         | 30%             |
| \$8,000.00–9,999.99         | 40%             |
| \$10,000.00–11,999.99       | 42%             |
| \$12,200.00–13,999.99       | 44%             |
| \$14,000.00 or greater      | 45%             |

|   | LAST_NAME | SALARY | TAX_RATE |
|---|-----------|--------|----------|
| 1 | Zlotkey   | 10500  | 0.42     |
| 2 | Abel      | 11000  | 0.42     |
| 3 | Taylor    | 8600   | 0.4      |

# Summary

In this lesson, you should have learned how to:

- Perform calculations on data using functions
- Modify individual data items using functions
- Manipulate output for groups of rows using functions
- Alter date formats for display using functions
- Convert column data types using functions
- Use `NVL` functions
- Use IF-THEN-ELSE logic



Copyright © 2009, Oracle. All rights reserved.

## Summary

Single-row functions can be nested to any level. Single-row functions can manipulate the following:

- Character data: `LOWER`, `UPPER`, `INITCAP`, `CONCAT`, `SUBSTR`, `INSTR`, `LENGTH`
- Number data: `ROUND`, `TRUNC`, `MOD`
- Date data: `MONTHS_BETWEEN`, `ADD_MONTHS`, `NEXT_DAY`, `LAST_DAY`, `ROUND`, `TRUNC`

Remember the following:

- Date values can also use arithmetic operators.
- Conversion functions can convert character, date, and numeric values: `TO_CHAR`, `TO_DATE`, `TO_NUMBER`
- There are several functions that pertain to nulls, including `NVL`, `NVL2`, `NULLIF`, and `COALESCE`.
- IF-THEN-ELSE logic can be applied within a SQL statement by using the `CASE` expression or the `DECODE` function.

## **SYSDATE and DUAL**

`SYSDATE` is a date function that returns the current date and time. It is customary to select `SYSDATE` from a dummy table called `DUAL`.

## Practice 3: Overview of Part 2

This practice covers the following topics:

- Creating queries that require the use of numeric, character, and date functions
- Using concatenation with functions
- Writing non-case-sensitive queries to test the usefulness of character functions
- Performing calculations of years and months of service for an employee
- Determining the review date for an employee



Copyright © 2009, Oracle. All rights reserved.

### Practice 3: Overview of Part 2

Part 2 of this lesson's practice provides a variety of exercises that use the different functions that are available for character, number, and date data types. For Part 2, complete exercises 7–14.

Remember that for nested functions, the results are evaluated from the innermost function to the outermost function.

## Practice 3

### Part 1

1. Write a query to display the current date. Label the column Date.

| Date        |
|-------------|
| 1 06-NOV-08 |

2. The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary. Place your SQL statement in a text file named lab\_03\_02.sql.
3. Run your query in the lab\_03\_02.sql file.

| EMPLOYEE_ID | LAST_NAME | SALARY | New Salary |
|-------------|-----------|--------|------------|
| 1           | Whalen    | 4400   | 5082       |
| 2           | Hartstein | 13000  | 15015      |
| 3           | Fay       | 6000   | 6930       |
| 4           | Higgins   | 12000  | 13860      |
| 5           | Gietz     | 8300   | 9587       |

...

|    |       |      |      |
|----|-------|------|------|
| 20 | Grant | 7000 | 8085 |
|----|-------|------|------|

4. Modify your lab\_03\_02.sql query to add a column that subtracts the old salary from the new salary. Label the column Increase. Save the contents of the file as lab\_03\_04.sql. Run the revised query.

| EMPLOYEE_ID | LAST_NAME | SALARY | New Salary | Increase |
|-------------|-----------|--------|------------|----------|
| 1           | Whalen    | 4400   | 5082       | 682      |
| 2           | Hartstein | 13000  | 15015      | 2015     |
| 3           | Fay       | 6000   | 6930       | 930      |
| 4           | Higgins   | 12000  | 13860      | 1860     |
| 5           | Gietz     | 8300   | 9587       | 1287     |
| 6           | King      | 24000  | 27720      | 3720     |
| 7           | Kochhar   | 17000  | 19635      | 2635     |
| 8           | De Haan   | 17000  | 19635      | 2635     |

...

|    |       |      |      |      |
|----|-------|------|------|------|
| 20 | Grant | 7000 | 8085 | 1085 |
|----|-------|------|------|------|

### Practice 3 (continued)

5. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters *J*, *A*, or *M*. Give each column an appropriate label. Sort the results by the last names of the employees.

|   | Name    | Length |
|---|---------|--------|
| 1 | Abel    | 4      |
| 2 | Matos   | 5      |
| 3 | Mourgos | 7      |

Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters *H* when prompted for a letter, the output should show all employees whose last name starts with the letter *H*.

|   | Name      | Length |
|---|-----------|--------|
| 1 | Hartstein | 9      |
| 2 | Higgins   | 7      |
| 3 | Hunold    | 6      |

6. The HR department wants to find the duration of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column *MONTHS\_WORKED*. Order your results by the number of months employed. Round the number of months up to the closest whole number.
- Note:** Your results will differ.

|    | LAST_NAME | MONTHS_WORKED |
|----|-----------|---------------|
| 1  | Zlotkey   | 105           |
| 2  | Mourgos   | 108           |
| 3  | Grant     | 113           |
| 4  | Lorentz   | 117           |
| 5  | Vargas    | 124           |
| 6  | Taylor    | 127           |
| 7  | Matos     | 128           |
| 8  | Fay       | 135           |
| 9  | Davies    | 141           |
| 10 | Abel      | 150           |

...

|         |     |
|---------|-----|
| 20 King | 257 |
|---------|-----|

## Practice 3 (continued)

### Part 2

7. Create a report that produces the following for each employee:

<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

| Dream Salaries |                                                            |
|----------------|------------------------------------------------------------|
| 1              | Whalen earns \$4,400.00 monthly but wants \$13,200.00.     |
| 2              | Hartstein earns \$13,000.00 monthly but wants \$39,000.00. |
| 3              | Fay earns \$6,000.00 monthly but wants \$18,000.00.        |
| 4              | Higgins earns \$12,000.00 monthly but wants \$36,000.00.   |

...

|    |                                                          |
|----|----------------------------------------------------------|
| 17 | Zlotkey earns \$10,500.00 monthly but wants \$31,500.00. |
| 18 | Abel earns \$11,000.00 monthly but wants \$33,000.00.    |
| 19 | Taylor earns \$8,600.00 monthly but wants \$25,800.00.   |
| 20 | Grant earns \$7,000.00 monthly but wants \$21,000.00.    |

If you have time, complete the following exercises:

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the \$ symbol. Label the column SALARY.

|   | LAST_NAME | SALARY                            |
|---|-----------|-----------------------------------|
| 1 | Whalen    | \$\$\$\$\$\$\$\$\$\$\$\$\$\$4400  |
| 2 | Hartstein | \$\$\$\$\$\$\$\$\$\$\$\$\$\$13000 |
| 3 | Fay       | \$\$\$\$\$\$\$\$\$\$\$\$\$\$6000  |
| 4 | Higgins   | \$\$\$\$\$\$\$\$\$\$\$\$\$\$12000 |
| 5 | Gietz     | \$\$\$\$\$\$\$\$\$\$\$\$\$\$8300  |
| 6 | King      | \$\$\$\$\$\$\$\$\$\$\$\$\$\$24000 |
| 7 | Kochhar   | \$\$\$\$\$\$\$\$\$\$\$\$\$\$17000 |

...

|    |         |                                   |
|----|---------|-----------------------------------|
| 16 | Vargas  | \$\$\$\$\$\$\$\$\$\$\$\$\$\$2500  |
| 17 | Zlotkey | \$\$\$\$\$\$\$\$\$\$\$\$\$\$10500 |
| 18 | Abel    | \$\$\$\$\$\$\$\$\$\$\$\$\$\$11000 |
| 19 | Taylor  | \$\$\$\$\$\$\$\$\$\$\$\$\$\$8600  |
| 20 | Grant   | \$\$\$\$\$\$\$\$\$\$\$\$\$\$7000  |

### Practice 3 (continued)

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

|   | LAST_NAME | HIRE_DATE | REVIEW                                     |
|---|-----------|-----------|--------------------------------------------|
| 1 | Whalen    | 17-SEP-87 | Monday, the Twenty-First of March, 1988    |
| 2 | Hartstein | 17-FEB-96 | Monday, the Nineteenth of August, 1996     |
| 3 | Fay       | 17-AUG-97 | Monday, the Twenty-Third of February, 1998 |
| 4 | Higgins   | 07-JUN-94 | Monday, the Twelfth of December, 1994      |
| 5 | Gietz     | 07-JUN-94 | Monday, the Twelfth of December, 1994      |
| 6 | King      | 17-JUN-87 | Monday, the Twenty-First of December, 1987 |

• • •

|    |         |           |                                              |
|----|---------|-----------|----------------------------------------------|
| 17 | Zlotkey | 29-JAN-00 | Monday, the Thirty-First of July, 2000       |
| 18 | Abel    | 11-MAY-96 | Monday, the Eighteenth of November, 1996     |
| 19 | Taylor  | 24-MAR-98 | Monday, the Twenty-Eighth of September, 1998 |
| 20 | Grant   | 24-MAY-99 | Monday, the Twenty-Ninth of November, 1999   |

10. Display the last name, hire date, and day of the week on which an employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

|   | LAST_NAME | HIRE_DATE | DAY       |
|---|-----------|-----------|-----------|
| 1 | Grant     | 24-MAY-99 | MONDAY    |
| 2 | Ernst     | 21-MAY-91 | TUESDAY   |
| 3 | Taylor    | 24-MAR-98 | TUESDAY   |
| 4 | Rajs      | 17-OCT-95 | TUESDAY   |
| 5 | Mourgos   | 16-NOV-99 | TUESDAY   |
| 6 | Gietz     | 07-JUN-94 | TUESDAY   |
| 7 | Higgins   | 07-JUN-94 | TUESDAY   |
| 8 | De Haan   | 13-JAN-93 | WEDNESDAY |

• • •

|    |           |           |          |
|----|-----------|-----------|----------|
| 16 | Zlotkey   | 29-JAN-00 | SATURDAY |
| 17 | Hartstein | 17-FEB-96 | SATURDAY |
| 18 | Lorentz   | 07-FEB-99 | SUNDAY   |
| 19 | Matos     | 15-MAR-98 | SUNDAY   |
| 20 | Fay       | 17-AUG-97 | SUNDAY   |

### Practice 3 (continued)

If you want an extra challenge, complete the following exercises:

11. Create a query that displays the employees' last names and commission amounts. If an employee does not earn a commission, show "No Commission." Label the column COMM.

| LAST_NAME   | COMM          |
|-------------|---------------|
| 1 Whalen    | No Commission |
| 2 Hartstein | No Commission |
| 3 Fay       | No Commission |

...

|            |               |
|------------|---------------|
| 14 Davies  | No Commission |
| 15 Matos   | No Commission |
| 16 Vargas  | No Commission |
| 17 Zlotkey | .2            |
| 18 Abel    | .3            |
| 19 Taylor  | .2            |
| 20 Grant   | .15           |

12. Create a query that displays the first eight characters of the employees' last names and indicates the amounts of their salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES\_AND\_THEIR\_SALARIES.

| EMPLOYEES_AND_THEIR_SALARIES |
|------------------------------|
| 1 King *****                 |
| 2 Kochhar *****              |
| 3 De Haan *****              |
| 4 Hartstei *****             |
| 5 Higgins *****              |
| 6 Abel *****                 |
| 7 Zlotkey *****              |

...

|           |     |
|-----------|-----|
| 17 Rajs   | *** |
| 18 Davies | *** |
| 19 Matos  | **  |
| 20 Vargas | **  |

### Practice 3 (continued)

13. Using the DECODE function, write a query that displays the grade of all employees based on the value of the JOB\_ID column, using the following data:

| <i>Job</i>        | <i>Grade</i> |
|-------------------|--------------|
| AD_PRES           | A            |
| ST_MAN            | B            |
| IT_PROG           | C            |
| SA_REP            | D            |
| ST_CLERK          | E            |
| None of the above | 0            |

| JOB_ID       | GRADE |
|--------------|-------|
| 1 AC_ACCOUNT | 0     |
| 2 AC_MGR     | 0     |
| 3 AD_ASST    | 0     |
| 4 AD_PRES    | A     |
| 5 AD_VP      | 0     |
| 6 AD_VP      | 0     |
| 7 IT_PROG    | C     |
| 8 IT_PROG    | C     |
| 9 IT_PROG    | C     |
| 10 MK_MAN    | 0     |
| 11 MK_REP    | 0     |
| 12 SA_MAN    | 0     |
| 13 SA_REP    | D     |
| 14 SA_REP    | D     |
| 15 SA_REP    | D     |
| 16 ST_CLERK  | E     |
| 17 ST_CLERK  | E     |
| 18 ST_CLERK  | E     |
| 19 ST_CLERK  | E     |
| 20 ST_MAN    | B     |

14. Rewrite the statement in the preceding exercise using the CASE syntax.

Oracle Internal & Oracle Academy Use Only

# **Reporting Aggregated Data Using the Group Functions**

Copyright © 2009, Oracle. All rights reserved.

**ORACLE®**

# Objectives

After completing this lesson, you should be able to do the following:

- Identify the available group functions
- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause



Copyright © 2009, Oracle. All rights reserved.

## Objectives

This lesson further addresses functions. It focuses on obtaining summary information (such as averages) for groups of rows. It discusses how to group rows in a table into smaller sets and how to specify search criteria for groups of rows.

# What Are Group Functions?

Group functions operate on sets of rows to give one result per group.

EMPLOYEES

|     | DEPARTMENT_ID | SALARY |
|-----|---------------|--------|
| 1   | 10            | 4400   |
| 2   | 20            | 13000  |
| 3   | 20            | 6000   |
| 4   | 110           | 12000  |
| 5   | 110           | 8300   |
| 6   | 90            | 24000  |
| 7   | 90            | 17000  |
| 8   | 90            | 17000  |
| 9   | 60            | 9000   |
| 10  | 60            | 6000   |
| 11  | 60            | 4200   |
| 12  | 50            | 5800   |
| 13  | 50            | 3500   |
| 14  | 50            | 3100   |
| 15  | 50            | 2600   |
| ... |               |        |

Maximum salary in  
EMPLOYEES table

|   | MAX(SALARY) |
|---|-------------|
| 1 | 24000       |

ORACLE

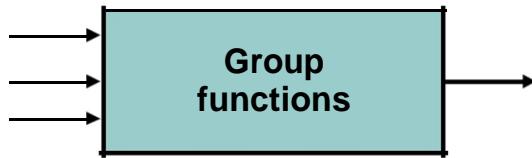
Copyright © 2009, Oracle. All rights reserved.

## Group Functions

Unlike single-row functions, group functions operate on sets of rows to give one result per group. These sets may comprise the entire table or the table split into groups.

# Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Types of Group Functions

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

| Function                                               | Description                                                                                                                                          |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| AVG ( [DISTINCT   <b>ALL</b> ] <i>n</i> )              | Average value of <i>n</i> , ignoring null values                                                                                                     |
| COUNT ( { *   [DISTINCT   <b>ALL</b> ] <i>expr</i> } ) | Number of rows, where <i>expr</i> evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls) |
| MAX ( [DISTINCT   <b>ALL</b> ] <i>expr</i> )           | Maximum value of <i>expr</i> , ignoring null values                                                                                                  |
| MIN ( [DISTINCT   <b>ALL</b> ] <i>expr</i> )           | Minimum value of <i>expr</i> , ignoring null values                                                                                                  |
| <b>STDDEV ( [DISTINCT   <b>ALL</b>] <i>x</i> )</b>     | Standard deviation of <i>n</i> , ignoring null values                                                                                                |
| SUM ( [DISTINCT   <b>ALL</b> ] <i>n</i> )              | Sum values of <i>n</i> , ignoring null values                                                                                                        |
| VARIANCE ( [DISTINCT   <b>ALL</b> ] <i>x</i> )         | Variance of <i>n</i> , ignoring null values                                                                                                          |

## Group Functions: Syntax

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column]
[ORDER BY column];
```



Copyright © 2009, Oracle. All rights reserved.

### Guidelines for Using Group Functions

- DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value, including duplicates. The default is ALL and therefore does not need to be specified.
- The data types for the functions with an expr argument may be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions ignore null values. To substitute a value for null values, use the NVL, NVL2, or COALESCE functions.

## Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
 MIN(salary), SUM(salary)
 FROM employees
 WHERE job_id LIKE '%REP%';
```

|   | AVG(SALARY) | MAX(SALARY) | MIN(SALARY) | SUM(SALARY) |
|---|-------------|-------------|-------------|-------------|
| 1 | 8150        | 11000       | 6000        | 32600       |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the Group Functions

You can use AVG, SUM, MIN, and MAX functions against columns that can store numeric data. The example in the slide displays the average, highest, lowest, and sum of monthly salaries for all sales representatives.

## Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

| MIN(HIRE_DATE) | MAX(HIRE_DATE) |
|----------------|----------------|
| 1 17-JUN-87    | 29-JAN-00      |



Copyright © 2009, Oracle. All rights reserved.

## Using the Group Functions (continued)

You can use the MAX and MIN functions for numeric, character, and date data types. The slide example displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetized list of all employees:

```
SELECT MIN(last_name), MAX(last_name)
FROM employees;
```

| MIN(LAST_NAME) | MAX(LAST_NAME) |
|----------------|----------------|
| Abel           | Zlotkey        |

**Note:** The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types. MAX and MIN cannot be used with LOB or LONG data types.

# Using the COUNT Function

COUNT (\*) returns the number of rows in a table:

1

```
SELECT COUNT(*)
FROM employees
WHERE department_id = 50;
```

|   | COUNT(*) |
|---|----------|
| 1 | 5        |

COUNT (expr) returns the number of rows with non-null values for expr:

2

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

|   | COUNT(COMMISSION_PCT) |
|---|-----------------------|
| 1 | 3                     |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## COUNT Function

The COUNT function has three formats:

- COUNT (\*)
- COUNT (expr)
- COUNT (DISTINCT expr)

COUNT (\*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns. If a WHERE clause is included in the SELECT statement, COUNT (\*) returns the number of rows that satisfy the condition in the WHERE clause.

In contrast, COUNT (expr) returns the number of non-null values that are in the column identified by expr.

COUNT (DISTINCT expr) returns the number of unique, non-null values that are in the column identified by expr.

## Examples

1. The slide example displays the number of employees in department 50.
2. The slide example displays the number of employees in department 80 who can earn a commission.

## Using the DISTINCT Keyword

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the EMPLOYEES table:

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

|   | COUNT(DISTINCTDEPARTMENT_ID) |
|---|------------------------------|
| 1 | 7                            |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DISTINCT Keyword

Use the DISTINCT keyword to suppress the counting of any duplicate values in a column.

The example in the slide displays the number of distinct department values that are in the EMPLOYEES table.

# Group Functions and Null Values

Group functions ignore null values in the column:

1

```
SELECT AVG(commission_pct)
FROM employees;
```

|   | AVG(COMMISSION_PCT) |
|---|---------------------|
| 1 | 0.2125              |

The NVL function forces group functions to include null values:

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

|   | AVG(NVL(COMMISSION_PCT,0)) |
|---|----------------------------|
| 1 | 0.0425                     |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Group Functions and Null Values

All group functions ignore null values in the column.

The NVL function forces group functions to include null values.

### Examples

1. The average is calculated based on *only* those rows in the table where a valid value is stored in the COMMISSION\_PCT column. The average is calculated as the total commission that is paid to all employees divided by the number of employees receiving a commission (four).
2. The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the COMMISSION\_PCT column. The average is calculated as the total commission that is paid to all employees divided by the total number of employees in the company (20).

# Creating Groups of Data

## EMPLOYEES

|    | DEPARTMENT_ID | SALARY |
|----|---------------|--------|
| 1  | 10            | 4400   |
| 2  | 20            | 13000  |
| 3  | 20            | 6000   |
| 4  | 50            | 2500   |
| 5  | 50            | 2600   |
| 6  | 50            | 3100   |
| 7  | 50            | 3500   |
| 8  | 50            | 5800   |
| 9  | 60            | 9000   |
| 10 | 60            | 6000   |
| 11 | 60            | 4200   |
| 12 | 80            | 11000  |
| 13 | 80            | 8600   |
| 14 | 80            | 10500  |
| 15 | 90            | 17000  |
| 16 | 90            | 24000  |
| 17 | 90            | 17000  |
| 18 | 110           | 8300   |
| 19 | 110           | 12000  |
| 20 | (null)        | 7000   |

4400

9500

3500

6400

10033

19333

10150

7000

Average  
salary in the  
EMPLOYEES  
table for each  
department

|   | DEPARTMENT_ID | AVG(SALARY)          |
|---|---------------|----------------------|
| 1 | 10            | 4400                 |
| 2 | 20            | 9500                 |
| 3 | 50            | 3500                 |
| 4 | 60            | 6400                 |
| 5 | 80            | 10033.33333333333... |
| 6 | 90            | 19333.33333333333... |
| 7 | 110           | 10150                |
| 8 | (null)        | 7000                 |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Creating Groups of Data

Until this point in our discussion, all group functions have treated the table as one large group of information.

At times, however, you need to divide the table of information into smaller groups. You can do this by using the GROUP BY clause.

## Creating Groups of Data: GROUP BY Clause Syntax

```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

You can divide rows in a table into smaller groups by using the GROUP BY clause.



Copyright © 2009, Oracle. All rights reserved.

### GROUP BY Clause

You can use the GROUP BY clause to divide the rows in a table into groups. You can then use the group functions to return summary information for each group.

In the syntax:

*group\_by\_expression* specifies columns whose values determine the basis for grouping rows

### Guidelines

- If you include a group function in a SELECT clause, you cannot select individual results as well, *unless* the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column list in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the *columns* in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.

## Using the GROUP BY Clause

All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

|   | DEPARTMENT_ID | Avg(SALARY)          |
|---|---------------|----------------------|
| 1 | (null)        | 7000                 |
| 2 | 20            | 9500                 |
| 3 | 90            | 19333.33333333333... |
| 4 | 110           | 10150                |
| 5 | 50            | 3500                 |
| 6 | 80            | 10033.33333333333... |
| 7 | 10            | 4400                 |
| 8 | 60            | 6400                 |

Copyright © 2009, Oracle. All rights reserved.

### Using the GROUP BY Clause

When using the GROUP BY clause, make sure that all columns in the SELECT list that are not group functions are included in the GROUP BY clause. The example in the slide displays the department number and the average salary for each department. Here is how this SELECT statement, containing a GROUP BY clause, is evaluated:

- The SELECT clause specifies the columns to be retrieved, as follows:
  - Department number column in the EMPLOYEES table
  - The average of all the salaries in the group that you specified in the GROUP BY clause
- The FROM clause specifies the tables that the database must access: the EMPLOYEES table.
- The WHERE clause specifies the rows to be retrieved. Because there is no WHERE clause, all rows are retrieved by default.
- The GROUP BY clause specifies how the rows should be grouped. The rows are grouped by department number, so the AVG function that is applied to the salary column calculates the *average salary for each department*.

## Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

```
SELECT AVG(salary)
FROM employees
GROUP BY department_id ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## **Using the GROUP BY Clause (continued)**

The GROUP BY column does not have to be in the SELECT clause. For example, the SELECT statement in the slide displays the average salaries for each department without displaying the respective department numbers. Without the department numbers, however, the results do not look meaningful.

You can use the group function in the ORDER BY clause:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
ORDER BY AVG(salary);
```

|   | DEPARTMENT_ID | Avg(Salary)          |
|---|---------------|----------------------|
| 1 | 50            | 35000                |
| 2 | 10            | 44000                |
| 3 | 60            | 64000                |
| 4 | (null)        | 70000                |
| 5 | 20            | 95000                |
| 6 | 80            | 100333.3333333333... |
| 7 | 110           | 101500               |
| 8 | 90            | 193333.3333333333... |

# Grouping by More Than One Column

**EMPLOYEES**

|    | DEPARTMENT_ID | JOB_ID     | SALARY |
|----|---------------|------------|--------|
| 1  | 10            | AD_ASST    | 4400   |
| 2  | 20            | MK_MAN     | 13000  |
| 3  | 20            | MK_REP     | 6000   |
| 4  | 50            | ST_CLERK   | 2500   |
| 5  | 50            | ST_CLERK   | 2600   |
| 6  | 50            | ST_CLERK   | 3100   |
| 7  | 50            | ST_CLERK   | 3500   |
| 8  | 50            | ST_MAN     | 5800   |
| 9  | 60            | IT_PROG    | 9000   |
| 10 | 60            | IT_PROG    | 6000   |
| 11 | 60            | IT_PROG    | 4200   |
| 12 | 80            | SA REP     | 11000  |
| 13 | 80            | SA REP     | 8600   |
| 14 | 80            | SA MAN     | 10500  |
| 15 | 90            | AD VP      | 17000  |
| 16 | 90            | AD PRES    | 24000  |
| 17 | 90            | AD VP      | 17000  |
| 18 | 110           | AC ACCOUNT | 8300   |
| 19 | 110           | AC MGR     | 12000  |
| 20 | (null)        | SA REP     | 7000   |

Add the salaries in the EMPLOYEES table for each job, grouped by department

|    | DEPARTMENT_ID | JOB_ID     | SUM(SALARY) |
|----|---------------|------------|-------------|
| 1  | 10            | AD_ASST    | 4400        |
| 2  | 20            | MK_MAN     | 13000       |
| 3  | 20            | MK_REP     | 6000        |
| 4  | 50            | ST_CLERK   | 11700       |
| 5  | 50            | ST_MAN     | 5800        |
| 6  | 60            | IT_PROG    | 19200       |
| 7  | 80            | SA_MAN     | 10500       |
| 8  | 80            | SA REP     | 19600       |
| 9  | 90            | AD PRES    | 24000       |
| 10 | 90            | AD VP      | 34000       |
| 11 | 110           | AC ACCOUNT | 8300        |
| 12 | 110           | AC MGR     | 12000       |
| 13 | (null)        | SA REP     | 7000        |

Copyright © 2009, Oracle. All rights reserved.

## Groups Within Groups



Sometimes you need to see results for groups within groups. The slide shows a report that displays the total salary that is paid to each job title in each department.

The EMPLOYEES table is grouped first by department number and then by job title within that grouping. For example, the four stock clerks in department 50 are grouped together, and a single result (total salary) is produced for all stock clerks in the group.

## Using the GROUP BY Clause on Multiple Columns

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id ;
```

|    | DEPT_ID | JOB_ID     | SUM(SALARY) |
|----|---------|------------|-------------|
| 1  | 110     | AC_ACCOUNT | 8300        |
| 2  | 90      | AD_VP      | 34000       |
| 3  | 50      | ST_CLERK   | 11700       |
| 4  | 80      | SA_REP     | 19600       |
| 5  | 110     | AC_MGR     | 12000       |
| 6  | 50      | ST_MAN     | 5800        |
| 7  | 80      | SA_MAN     | 10500       |
| 8  | 20      | MK_MAN     | 13000       |
| 9  | 90      | AD_PRES    | 24000       |
| 10 | 60      | IT_PROG    | 19200       |
| 11 | (null)  | SA_REP     | 7000        |
| 12 | 10      | AD_ASST    | 4400        |
| 13 | 20      | MK_REP     | 6000        |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Groups Within Groups (continued)

You can return summary results for groups and subgroups by listing more than one GROUP BY column. You can determine the default sort order of the results by the order of the columns in the GROUP BY clause. In the slide example, the SELECT statement containing a GROUP BY clause is evaluated as follows:

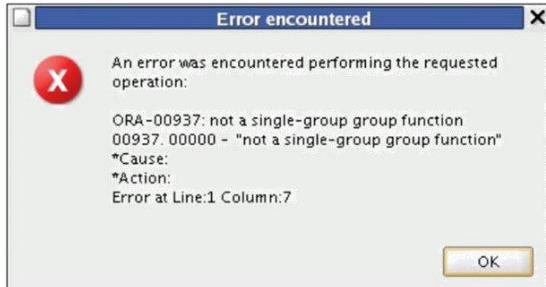
- The SELECT clause specifies the column to be retrieved:
  - Department number in the EMPLOYEES table
  - Job ID in the EMPLOYEES table
  - The sum of all the salaries in the group that you specified in the GROUP BY clause
- The FROM clause specifies the tables that the database must access: the EMPLOYEES table.
- The GROUP BY clause specifies how you must group the rows:
  - First, the rows are grouped by department number.
  - Second, the rows are grouped by job ID in the department number groups.

So the SUM function is applied to the salary column for all job IDs in each department number group.

## Illegal Queries Using Group Functions

Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause:

```
SELECT department_id, COUNT(last_name)
FROM employees;
```



Column missing in the `GROUP BY` clause

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Illegal Queries Using Group Functions

Whenever you use a mixture of individual items (`DEPARTMENT_ID`) and group functions (`COUNT`) in the same `SELECT` statement, you must include a `GROUP BY` clause that specifies the individual items (in this case, `DEPARTMENT_ID`). If the `GROUP BY` clause is missing, then the error message “not a single-group group function” appears. You can correct the error in the slide by adding the `GROUP BY` clause:

```
SELECT department_id, count(last_name)
FROM employees
GROUP BY department_id;
```

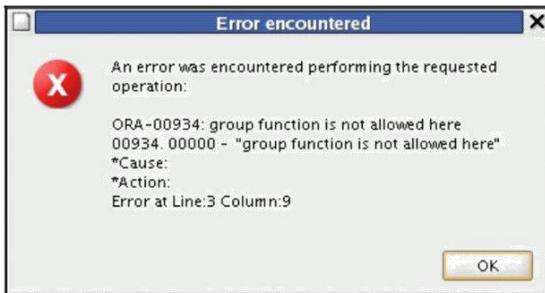
|   | DEPARTMENT_ID | COUNT(LAST_NAME) |
|---|---------------|------------------|
| 1 | (null)        | 1                |
| 2 | 20            | 2                |
| 3 | 90            | 3                |
| 4 | 110           | 2                |
| 5 | 50            | 5                |
| 6 | 80            | 3                |
| 7 | 10            | 1                |
| 8 | 60            | 3                |

Any column or expression in the `SELECT` list that is not an aggregate function must be in the `GROUP BY` clause.

## Illegal Queries Using Group Functions

- You cannot use the WHERE clause to restrict groups.
- You use the HAVING clause to restrict groups.
- You cannot use group functions in the WHERE clause.

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary) > 8000
GROUP BY department_id;
```



Cannot use the WHERE clause  
to restrict groups

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Illegal Queries Using Group Functions (continued)

The WHERE clause cannot be used to restrict groups. The SELECT statement in the slide example results in an error because it uses the WHERE clause to restrict the display of average salaries of those departments that have an average salary greater than \$8,000.

You can correct the error in the example by using the HAVING clause to restrict groups:

```
SELECT department_id, AVG(salary)
FROM employees
HAVING AVG(salary) > 8000
GROUP BY department_id;
```

|   | DEPARTMENT_ID | AVG(SALARY)             |
|---|---------------|-------------------------|
| 1 | 20            | 9500                    |
| 2 | 90            | 19333.33333333333333... |
| 3 | 110           | 10150                   |
| 4 | 80            | 10033.33333333333333... |

# Restricting Group Results

**EMPLOYEES**

|     | DEPARTMENT_ID | SALARY |
|-----|---------------|--------|
| 1   | 10            | 4400   |
| 2   | 20            | 13000  |
| 3   | 20            | 6000   |
| 4   | 110           | 12000  |
| 5   | 110           | 8300   |
| 6   | 90            | 24000  |
| 7   | 90            | 17000  |
| 8   | 90            | 17000  |
| 9   | 60            | 9000   |
| 10  | 60            | 6000   |
| 11  | 60            | 4200   |
| 12  | 50            | 5800   |
| 13  | 50            | 3500   |
| 14  | 50            | 3100   |
| 15  | 50            | 2600   |
| ... |               |        |

The maximum salary per department when it is greater than \$10,000

|   | DEPARTMENT_ID | MAX(SALARY) |
|---|---------------|-------------|
| 1 | 20            | 13000       |
|   | 80            | 11000       |
| 3 | 90            | 24000       |
| 4 | 110           | 12000       |

Copyright © 2009, Oracle. All rights reserved.

## Restricting Group Results



In the same way that you use the WHERE clause to restrict the rows that you select, you use the HAVING clause to restrict groups. To find the maximum salary in each of the departments that have a maximum salary greater than \$10,000, you need to do the following:

1. Find the average salary for each department by grouping by department number.
2. Restrict the groups to those departments with a maximum salary greater than \$10,000.

## Restricting Group Results with the HAVING Clause

When you use the HAVING clause, the Oracle server restricts groups as follows:

- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Restricting Group Results with the HAVING Clause

You use the HAVING clause to specify which groups are to be displayed, thus further restricting the groups on the basis of aggregate information.

In the syntax, *group\_condition* restricts the groups of rows returned to those groups for which the specified condition is true.

The Oracle server performs the following steps when you use the HAVING clause:

1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the HAVING clause are displayed.

The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because that is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

# Using the HAVING Clause

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000 ;
```

|   | DEPARTMENT_ID | MAX(SALARY) |
|---|---------------|-------------|
| 1 | 20            | 13000       |
| 2 | 90            | 24000       |
| 3 | 110           | 12000       |
| 4 | 80            | 11000       |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using the HAVING Clause

The slide example displays department numbers and maximum salaries for those departments with a maximum salary that is greater than \$10,000.

You can use the GROUP BY clause without using a group function in the SELECT list.

If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.

The following example displays the department numbers and average salaries for those departments with a maximum salary that is greater than \$10,000:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING max(salary)>10000;
```

|   | DEPARTMENT_ID | AVG(SALARY)                 |
|---|---------------|-----------------------------|
| 1 | 20            | 9500                        |
| 2 | 90            | 19333.333333333333333333... |
| 3 | 110           | 10150                       |
| 4 | 80            | 10033.333333333333333333... |

## Using the HAVING Clause

```
SELECT job_id, SUM(salary) PAYROLL
FROM employees
WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING SUM(salary) > 13000
ORDER BY SUM(salary);
```

| JOB_ID    | PAYROLL |
|-----------|---------|
| 1 IT_PROG | 19200   |
| 2 AD_PRES | 24000   |
| 3 AD_VP   | 34000   |



Copyright © 2009, Oracle. All rights reserved.

### Using the HAVING Clause (continued)

The slide example displays the job ID and total monthly salary for each job that has a total payroll exceeding \$13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

# Nesting Group Functions

Display the maximum average salary:

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

MAX(AVG(SALARY))

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Nesting Group Functions

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.

## Summary

In this lesson, you should have learned how to:

- Use the group functions COUNT, MAX, MIN, and AVG
- Write queries that use the GROUP BY clause
- Write queries that use the HAVING clause

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```



Copyright © 2009, Oracle. All rights reserved.

## Summary

Several group functions are available in SQL, such as the following:

AVG, COUNT, MAX, MIN, SUM, STDDEV, and VARIANCE

You can create subgroups by using the GROUP BY clause. Groups can be restricted using the HAVING clause.

Place the HAVING and GROUP BY clauses after the WHERE clause in a statement. The order of the HAVING and GROUP clauses following the WHERE clause is not important. Place the ORDER BY clause last.

The Oracle server evaluates the clauses in the following order:

1. If the statement contains a WHERE clause, the server establishes the candidate rows.
2. The server identifies the groups that are specified in the GROUP BY clause.
3. The HAVING clause further restricts result groups that do not meet the group criteria in the HAVING clause.

**Note:** For a complete list of the group functions, see *Oracle SQL Reference*.

## Practice 4: Overview

This practice covers the following topics:

- Writing queries that use the group functions
- Grouping by rows to achieve multiple results
- Restricting groups by using the HAVING clause



Copyright © 2009, Oracle. All rights reserved.

### Practice 4: Overview

After completing this practice, you should be familiar with using group functions and selecting groups of data.

## Practice 4

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.

True/False

2. Group functions include nulls in calculations.

True/False

3. The WHERE clause restricts rows before inclusion in a group calculation.

True/False

The HR department needs the following reports:

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Place your SQL statement in a text file named lab\_04\_04.sql.

|   | Maximum | Minimum | Sum    | Average |
|---|---------|---------|--------|---------|
| 1 | 24000   | 2500    | 175500 | 8775    |

5. Modify the query in lab\_04\_04.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab\_04\_04.sql as lab\_04\_05.sql. Run the statement in lab\_04\_05.sql.

|    | JOB_ID     | Maximum | Minimum | Sum   | Average |
|----|------------|---------|---------|-------|---------|
| 1  | AC_MGR     | 12000   | 12000   | 12000 | 12000   |
| 2  | AC_ACCOUNT | 8300    | 8300    | 8300  | 8300    |
| 3  | IT_PROG    | 9000    | 4200    | 19200 | 6400    |
| 4  | ST_MAN     | 5800    | 5800    | 5800  | 5800    |
| 5  | AD_ASST    | 4400    | 4400    | 4400  | 4400    |
| 6  | AD_VP      | 17000   | 17000   | 34000 | 17000   |
| 7  | MK_MAN     | 13000   | 13000   | 13000 | 13000   |
| 8  | SA_MAN     | 10500   | 10500   | 10500 | 10500   |
| 9  | MK_REP     | 6000    | 6000    | 6000  | 6000    |
| 10 | AD_PRES    | 24000   | 24000   | 24000 | 24000   |
| 11 | SA REP     | 11000   | 7000    | 26600 | 8867    |
| 12 | ST_CLERK   | 3500    | 2500    | 11700 | 2925    |

## Practice 4 (continued)

6. Write a query to display the number of people with the same job.

| JOB_ID       | COUNT(*) |
|--------------|----------|
| 1 AC_ACCOUNT | 1        |
| 2 AC_MGR     | 1        |
| 3 AD_ASST    | 1        |
| 4 AD PRES    | 1        |
| 5 AD_VP      | 2        |
| 6 IT_PROG    | 3        |
| 7 MK_MAN     | 1        |
| 8 MK_REP     | 1        |
| 9 SA_MAN     | 1        |
| 10 SA REP    | 3        |
| 11 ST_CLERK  | 4        |
| 12 ST_MAN    | 1        |

Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named `lab_04_06.sql`.

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER\_ID column to determine the number of managers.*

| Number of Managers |
|--------------------|
| 8                  |

8. Find the difference between the highest and the lowest salaries. Label the column DIFFERENCE.

| DIFFERENCE |
|------------|
| 21500      |

If you have time, complete the following exercises:

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is \$6,000 or less. Sort the output in descending order of salary.

| MANAGER_ID | MIN(SALARY) |
|------------|-------------|
| 1          | 9000        |
| 2          | 8300        |
| 3          | 7000        |

## Practice 4 (continued)

If you want an extra challenge, complete the following exercises:

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

|   | TOTAL | 1995 | 1996 | 1997 | 1998 |
|---|-------|------|------|------|------|
| 1 | 20    | 1    | 2    | 2    | 3    |

11. Create a matrix query to display the job, the salary for that job based on the department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

|    | Job        | Dept 20 | Dept 50 | Dept 80 | Dept 90 | Total |
|----|------------|---------|---------|---------|---------|-------|
| 1  | AC_MGR     | (null)  | (null)  | (null)  | (null)  | 12000 |
| 2  | AC_ACCOUNT | (null)  | (null)  | (null)  | (null)  | 8300  |
| 3  | IT_PROG    | (null)  | (null)  | (null)  | (null)  | 19200 |
| 4  | ST_MAN     | (null)  | 5800    | (null)  | (null)  | 5800  |
| 5  | AD_ASST    | (null)  | (null)  | (null)  | (null)  | 4400  |
| 6  | AD_VP      | (null)  | (null)  | (null)  | 34000   | 34000 |
| 7  | MK_MAN     | 13000   | (null)  | (null)  | (null)  | 13000 |
| 8  | SA_MAN     | (null)  | (null)  | 10500   | (null)  | 10500 |
| 9  | MK_REP     | 6000    | (null)  | (null)  | (null)  | 6000  |
| 10 | AD PRES    | (null)  | (null)  | (null)  | 24000   | 24000 |
| 11 | SA REP     | (null)  | (null)  | 19600   | (null)  | 26600 |
| 12 | ST_CLERK   | (null)  | 11700   | (null)  | (null)  | 11700 |



## **Displaying Data from Multiple Tables**

**ORACLE®**

Copyright © 2009, Oracle. All rights reserved.

# Objectives

After completing this lesson, you should be able to do the following:

- Write SELECT statements to access data from more than one table using equijoins and nonequijoins
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using outer joins
- Generate a Cartesian product of all rows from two or more tables



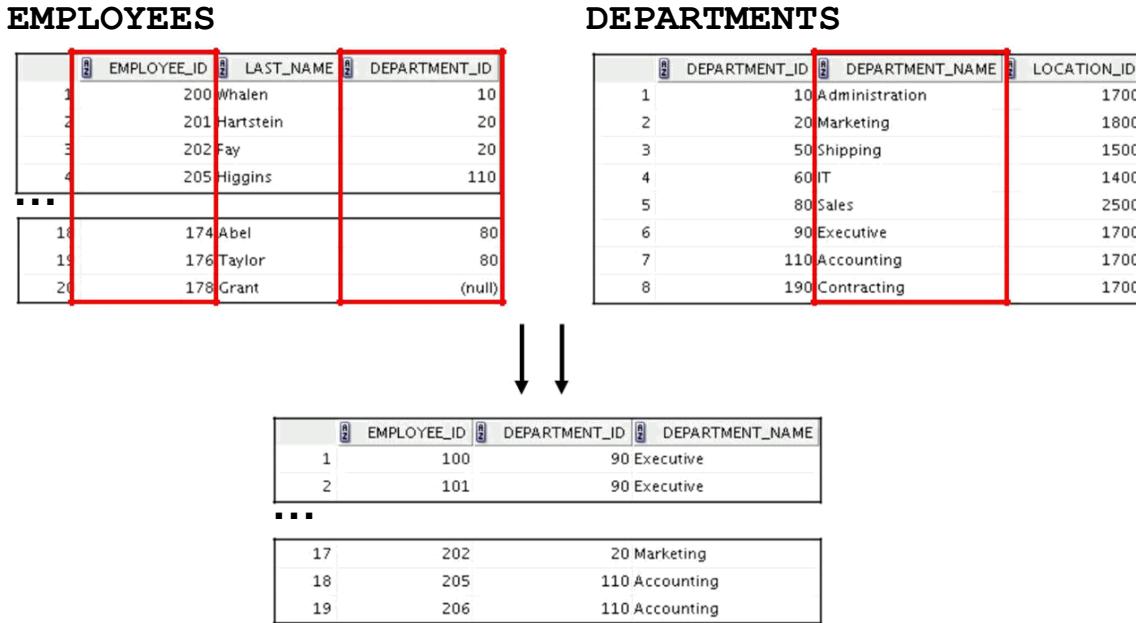
Copyright © 2009, Oracle. All rights reserved.

## Objectives

This lesson explains how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

**Note:** Information on joins is found in “SQL Queries and Subqueries: Joins” in *Oracle SQL Reference*.

# Obtaining Data from Multiple Tables



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Obtaining Data from Multiple Tables

Sometimes you need to use data from more than one table. In the slide example, the report displays data from two separate tables:

- Employee IDs exist in the EMPLOYEES table.
- Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
- Department names exist in the DEPARTMENTS table.

To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables and access data from both of them.

## Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Cross joins
- Natural joins
- USING clause
- Full (or two-sided) outer joins
- Arbitrary join conditions for outer joins



Copyright © 2009, Oracle. All rights reserved.

### Types of Joins

To join tables, you can use join syntax that is compliant with the SQL:1999 standard.

**Note:** Before the Oracle9*i* release, the join syntax was different from the ANSI standards. The SQL:1999-compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in prior releases. For detailed information about the proprietary join syntax, see Appendix C.

# Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT table1.column, table2.column
FROM table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
 ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
 ON (table1.column_name =
 table2.column_name)] | [CROSS JOIN table2];
```



Copyright © 2009, Oracle. All rights reserved.

## Defining Joins

In the syntax:

*table1.column* denotes the table and column from which data is retrieved  
NATURAL JOIN joins two tables based on the same column name  
JOIN *table* USING *column\_name* performs an equijoin based on the column name  
JOIN *table* ON *table1.column\_name* performs an equijoin based on the condition in the ON clause, = *table2.column\_name*  
*LEFT/RIGHT/FULL OUTER* is used to perform outer joins  
CROSS JOIN returns a Cartesian product from the two tables  
For more information, see “SELECT” in *Oracle SQL Reference*.

## Creating Natural Joins

- The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating Natural Joins

You can join tables automatically based on columns in the two tables that have matching data types and names. You do this by using the keywords NATURAL JOIN.

**Note:** The join can happen on only those columns that have the same names and data types in both tables. If the columns have the same name but different data types, then the NATURAL JOIN syntax causes an error.

## Retrieving Records with Natural Joins

```
SELECT department_id, department_name,
 location_id, city
 FROM departments
NATURAL JOIN locations ;
```

|   | DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID | CITY                |
|---|---------------|-----------------|-------------|---------------------|
| 1 | 60            | IT              | 1400        | Southlake           |
| 2 | 50            | Shipping        | 1500        | South San Francisco |
| 3 | 10            | Administration  | 1700        | Seattle             |
| 4 | 90            | Executive       | 1700        | Seattle             |
| 5 | 110           | Accounting      | 1700        | Seattle             |
| 6 | 190           | Contracting     | 1700        | Seattle             |
| 7 | 20            | Marketing       | 1800        | Toronto             |
| 8 | 80            | Sales           | 2500        | Oxford              |

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Retrieving Records with Natural Joins

In the example in the slide, the LOCATIONS table is joined to the DEPARTMENT table by the LOCATION\_ID column, which is the only column of the same name in both tables. If other common columns were present, the join would have used them all.

#### Natural Joins with a WHERE Clause

Additional restrictions on a natural join are implemented by using a WHERE clause. The following example limits the rows of output to those with a department ID equal to 20 or 50:

```
SELECT department_id, department_name,
 location_id, city
 FROM departments
NATURAL JOIN locations
 WHERE department_id IN (20, 50);
```

## Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, natural join can be applied by using the USING clause to specify the columns that should be used for an equijoin.
- Use the USING clause to match only one column when more than one column matches.
- Do not use a table name or alias in the referenced columns.
- The NATURAL JOIN and USING clauses are mutually exclusive.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### USING Clause

Natural joins use all columns with matching names and data types to join the tables. The USING clause can be used to specify only those columns that should be used for an equijoin. The columns that are referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement.

For example, the following statement is valid:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE location_id = 1400;
```

The following statement is invalid because the LOCATION\_ID is qualified in the WHERE clause:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE d.location_id = 1400;
ORA-25154: column part of USING clause cannot have qualifier
```

The same restriction also applies to NATURAL joins. Therefore, columns that have the same name in both tables must be used without any qualifiers.

# Joining Column Names

**EMPLOYEES**

|    | EMPLOYEE_ID | DEPARTMENT_ID |
|----|-------------|---------------|
| 1  | 200         | 10            |
| 2  | 201         | 20            |
| 3  | 202         | 20            |
| 4  | 205         | 110           |
| 5  | 206         | 110           |
| 6  | 100         | 90            |
| 7  | 101         | 90            |
| 8  | 102         | 90            |
| 9  | 103         | 60            |
| 10 | 104         | 60            |
| 11 | 107         | 60            |
| 12 | 124         | 50            |
| 13 | 141         | 50            |
| 14 | 142         | 50            |
| 15 | 143         | 50            |
| 16 | 144         | 50            |
| 17 | 149         | 80            |
| 18 | 174         | 80            |
| 19 | 176         | 80            |
| 20 | 178         | (null)        |

**DEPARTMENTS**

|    | DEPARTMENT_ID | DEPARTMENT_NAME |
|----|---------------|-----------------|
| 1  | 10            | Administration  |
| 2  | 20            | Marketing       |
| 3  | 20            | Marketing       |
| 4  | 50            | Shipping        |
| 5  | 50            | Shipping        |
| 6  | 50            | Shipping        |
| 7  | 50            | Shipping        |
| 8  | 50            | Shipping        |
| 9  | 60            | T               |
| 10 | 60            | T               |
| 11 | 60            | T               |
| 12 | 80            | Sales           |
| 13 | 80            | Sales           |
| 14 | 80            | Sales           |
| 15 | 90            | Executive       |
| 16 | 90            | Executive       |
| 17 | 90            | Executive       |
| 18 | 110           | Accounting      |
| 19 | 110           | Accounting      |

Foreign key

Primary key

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## The USING Clause for Equijoins

To determine an employee's department name, you compare the value in the DEPARTMENT\_ID column in the EMPLOYEES table with the DEPARTMENT\_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin*; that is, values in the DEPARTMENT\_ID column in both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

**Note:** Equijoins are also called *simple joins* or *inner joins*.

# Retrieving Records with the USING Clause

```
SELECT employees.employee_id, employees.last_name,
 departments.location_id, department_id
 FROM employees JOIN departments
USING (department_id) ;
```

|     | EMPLOYEE_ID | LAST_NAME | LOCATION_ID | DEPARTMENT_ID |
|-----|-------------|-----------|-------------|---------------|
| 1   | 200         | Whalen    | 1700        | 10            |
| 2   | 201         | Hartstein | 1800        | 20            |
| 3   | 202         | Fay       | 1800        | 20            |
| 4   | 205         | Higgins   | 1700        | 110           |
| 5   | 206         | Gietz     | 1700        | 110           |
| 6   | 100         | King      | 1700        | 90            |
| 7   | 101         | Kochhar   | 1700        | 90            |
| 8   | 102         | De Haan   | 1700        | 90            |
| 9   | 103         | Hunold    | 1400        | 60            |
| 10  | 104         | Ernst     | 1400        | 60            |
| ... |             |           |             |               |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Retrieving Records with the USING Clause

The slide example joins the DEPARTMENT\_ID column in the EMPLOYEES and DEPARTMENTS tables, and thus shows the location where an employee works.

## Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.
- Do not use aliases on columns that are identified in the USING clause and listed elsewhere in the SQL statement.



Copyright © 2009, Oracle. All rights reserved.

### Qualifying Ambiguous Column Names

You need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT\_ID column in the SELECT list could be from either the DEPARTMENTS table or the EMPLOYEES table. It is necessary to add the table prefix to execute your query:

```
SELECT employees.employee_id, employees.last_name,
 departments.department_id, departments.location_id
 FROM employees JOIN departments
 WHERE employees.department_id = departments.department_id;
```

If there are no common column names between the two tables, there is no need to qualify the columns. However, using the table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

**Note:** When joining with the USING clause, you cannot qualify a column that is used in the USING clause itself. Furthermore, if that column is used anywhere in the SQL statement, you cannot alias it.

# Using Table Aliases

- Use table aliases to simplify queries.
- Use table aliases to improve performance.

```
SELECT e.employee_id, e.last_name,
 d.location_id, department_id
 FROM employees e JOIN departments d
 USING (department_id) ;
```



Copyright © 2009, Oracle. All rights reserved.

## Using Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. You can use *table aliases* instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, therefore using less memory.

Notice how table aliases are identified in the FROM clause in the example. The table name is specified in full, followed by a space and then the table alias. The EMPLOYEES table has been given an alias of e, and the DEPARTMENTS table an alias of d.

### Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid for only the current SELECT statement.

## Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the `ON` clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The `ON` clause makes code easy to understand.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### **ON Clause**

Use the `ON` clause to specify a join condition. This lets you specify join conditions separate from any search or filter conditions in the `WHERE` clause.

## Retrieving Records with the ON Clause

```
SELECT e.employee_id, e.last_name, e.department_id,
 d.department_id, d.location_id
 FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id);
```

|     | EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID_1 | LOCATION_ID |
|-----|-------------|-----------|---------------|-----------------|-------------|
| 1   | 200         | Whalen    | 10            | 10              | 1700        |
| 2   | 201         | Hartstein | 20            | 20              | 1800        |
| 3   | 202         | Fay       | 20            | 20              | 1800        |
| 4   | 205         | Higgins   | 110           | 110             | 1700        |
| 5   | 206         | Gietz     | 110           | 110             | 1700        |
| 6   | 100         | King      | 90            | 90              | 1700        |
| 7   | 101         | Kochhar   | 90            | 90              | 1700        |
| 8   | 102         | De Haan   | 90            | 90              | 1700        |
| 9   | 103         | Hunold    | 60            | 60              | 1400        |
| 10  | 104         | Ernst     | 60            | 60              | 1400        |
| ... |             |           |               |                 |             |



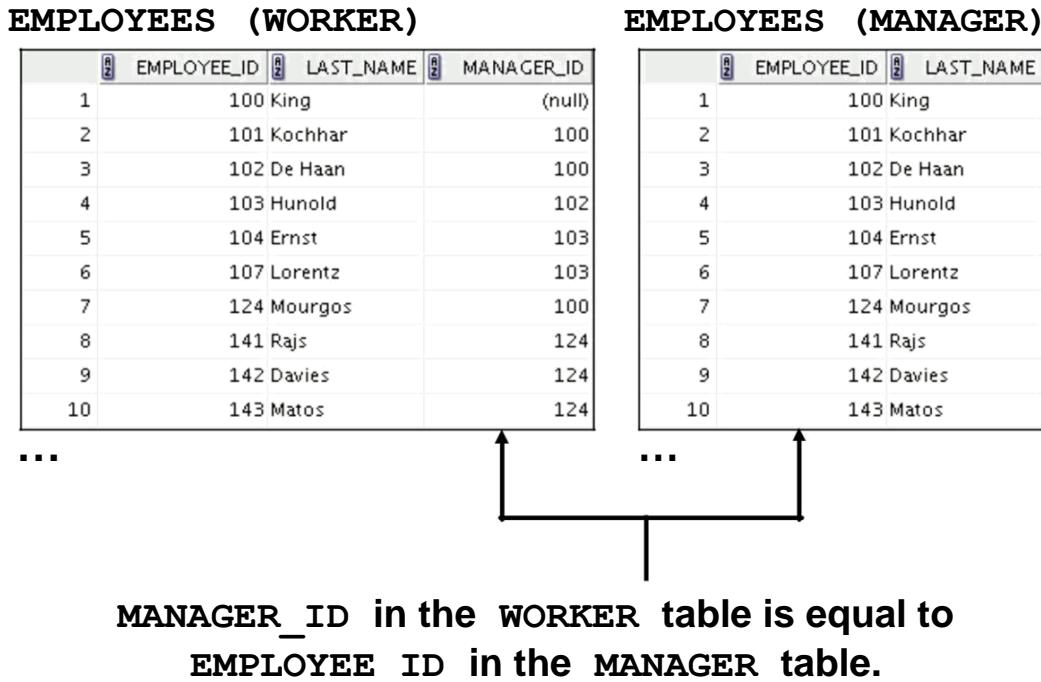
Copyright © 2009, Oracle. All rights reserved.

### Creating Joins with the ON Clause

In this example, the DEPARTMENT\_ID columns in the EMPLOYEES and DEPARTMENTS table are joined using the ON clause. Wherever a department ID in the EMPLOYEES table equals a department ID in the DEPARTMENTS table, the row is returned.

You can also use the ON clause to join columns that have different names.

## Self-Joins Using the ON Clause



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the **EMPLOYEES** table to itself, or perform a self join. For example, to find the name of Lorentz's manager, you need to:

- Find Lorentz in the **EMPLOYEES** table by looking at the **LAST\_NAME** column
- Find the manager number for Lorentz by looking at the **MANAGER\_ID** column. Lorentz's manager number is 103.
- Find the name of the manager with **EMPLOYEE\_ID** 103 by looking at the **LAST\_NAME** column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the **LAST\_NAME** column and **MANAGER\_ID** value of 103. The second time you look in the **EMPLOYEE\_ID** column to find 103 and the **LAST\_NAME** column to find Hunold.

## Self-Joins Using the ON Clause

```
SELECT e.last_name emp, m.last_name mgr
FROM employees e JOIN employees m
ON (e.manager_id = m.employee_id);
```

|     | EMP       | MGR       |
|-----|-----------|-----------|
| 1   | Abel      | Zlotkey   |
| 2   | Davies    | Mourgos   |
| 3   | De Haan   | King      |
| 4   | Ernst     | Hunold    |
| 5   | Fay       | Hartstein |
| 6   | Gietz     | Higgins   |
| 7   | Grant     | Zlotkey   |
| 8   | Hartstein | King      |
| ... |           |           |



Copyright © 2009, Oracle. All rights reserved.

### Joining a Table to Itself (continued)

The ON clause can also be used to join columns that have different names, within the same table or in a different table.

The example shown is a self-join of the EMPLOYEES table, based on the EMPLOYEE\_ID and MANAGER\_ID columns.

## Applying Additional Conditions to a Join

```
SELECT e.employee_id, e.last_name, e.department_id,
 d.department_id, d.location_id
 FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id)
 AND e.manager_id = 149;
```

|   | EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_ID_1 | LOCATION_ID |
|---|-------------|-----------|---------------|-----------------|-------------|
| 1 | 174         | Abel      | 80            | 80              | 2500        |
| 2 | 176         | Taylor    | 80            | 80              | 2500        |



Copyright © 2009, Oracle. All rights reserved.

### Applying Additional Conditions to a Join

You can apply additional conditions to the join.

The example shown performs a join on the EMPLOYEES and DEPARTMENTS tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the ON clause, you can add AND clauses. Alternatively, you can use a WHERE clause to apply additional conditions:

```
SELECT e.employee_id, e.last_name, e.department_id,
 d.department_id, d.location_id
 FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id)
 AND e.manager_id = 149;
```

## Creating Three-Way Joins with the ON Clause

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

|     | EMPLOYEE_ID | CITY                | DEPARTMENT_NAME |
|-----|-------------|---------------------|-----------------|
| 1   | 100         | Seattle             | Executive       |
| 2   | 101         | Seattle             | Executive       |
| 3   | 102         | Seattle             | Executive       |
| 4   | 103         | Southlake           | IT              |
| 5   | 104         | Southlake           | IT              |
| 6   | 107         | Southlake           | IT              |
| 7   | 124         | South San Francisco | Shipping        |
| 8   | 141         | South San Francisco | Shipping        |
| ... |             |                     |                 |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### Three-Way Joins

A three-way join is a join of three tables. In SQL:1999-compliant syntax, joins are performed from left to right. So, the first join to be performed is EMPLOYEES JOIN DEPARTMENTS. The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS. The second join condition can reference columns from all three tables.

## Nonequi joins

**EMPLOYEES**

|     | LAST_NAME | SALARY |
|-----|-----------|--------|
| 1   | Whalen    | 4400   |
| 2   | Hartstein | 13000  |
| 3   | Fay       | 6000   |
| 4   | Higgins   | 12000  |
| 5   | Gietz     | 8300   |
| 6   | King      | 24000  |
| 7   | Kochhar   | 17000  |
| 8   | De Haan   | 17000  |
| 9   | Hunold    | 9000   |
| 10  | Ernst     | 6000   |
| ... |           |        |

**JOB\_GRADES**

|   | GRADE_LEVEL | LOWEST_SAL | HIGHEST_SAL |
|---|-------------|------------|-------------|
| 1 | A           | 1000       | 2999        |
| 2 | B           | 3000       | 5999        |
| 3 | C           | 6000       | 9999        |
| 4 | D           | 10000      | 14999       |
| 5 | E           | 15000      | 24999       |
| 6 | F           | 25000      | 40000       |

← Salary in the EMPLOYEES

table must be between  
lowest salary and highest  
salary in the JOB\_GRADES  
table.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Nonequi joins

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the EMPLOYEES table and the JOB\_GRADES table is an example of a nonequijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST\_SALARY and HIGHEST\_SALARY columns of the JOB\_GRADES table. The relationship is obtained using an operator other than equality ( $=$ ).

# Retrieving Records with Nonequiijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM employees e JOIN job_grades j
ON e.salary
 BETWEEN j.lowest_sal AND j.highest_sal;
```

|     | LAST_NAME | SALARY | GRADE_LEVEL |
|-----|-----------|--------|-------------|
| 1   | Vargas    | 2500   | A           |
| 2   | Matos     | 2600   | A           |
| 3   | Davies    | 3100   | B           |
| 4   | Rajs      | 3500   | B           |
| 5   | Lorentz   | 4200   | B           |
| 6   | Whalen    | 4400   | B           |
| 7   | Mourgos   | 5800   | B           |
| 8   | Ernst     | 6000   | C           |
| ... |           |        |             |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Nonequiijoins (continued)

The slide example creates a nonequijoin to evaluate an employee's salary grade. The salary must be *between* any pair of the low and high salary ranges.

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

- None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits that are provided by the job grade table. That is, no employee earns less than the lowest value contained in the `LOWEST_SAL` column or more than the highest value contained in the `HIGHEST_SAL` column.

**Note:** Other conditions (such as `<=` and `>=`) can be used, but `BETWEEN` is the simplest. Remember to specify the low value first and the high value last when using `BETWEEN`.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

## Outer Joins

DEPARTMENTS

|   | DEPARTMENT_NAME | DEPARTMENT_ID |
|---|-----------------|---------------|
| 1 | Administration  | 10            |
| 2 | Marketing       | 20            |
| 3 | Shipping        | 50            |
| 4 | IT              | 60            |
| 5 | Sales           | 80            |
| 6 | Executive       | 90            |
| 7 | Accounting      | 110           |
| 8 | Contracting     | 190           |

EMPLOYEES

|    | DEPARTMENT_ID | LAST_NAME |
|----|---------------|-----------|
| 1  | 10            | Whalen    |
| 2  | 20            | Hartstein |
| 3  | 20            | Fay       |
| 4  | 110           | Higgins   |
| 5  | 110           | Gietz     |
| 6  | 90            | King      |
| 7  | 90            | Kochhar   |
| 8  | 90            | De Haan   |
| 9  | 60            | Hunold    |
| 10 | 60            | Ernst     |

...

There are no employees in department 190.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of EMPLOYEES and DEPARTMENTS tables, department ID 190 does not appear because there are no employees with that department ID recorded in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

To return the department record that does not have any employees, you can use an outer join.

## INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an inner join.
- A join between two tables that returns the results of the inner join as well as the unmatched rows from the left (or right) tables is called a left (or right) outer join.
- A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INNER Versus OUTER Joins

Joining tables with the NATURAL JOIN, USING, or ON clauses results in an inner join. Any unmatched rows are not displayed in the output. To return the unmatched rows, you can use an outer join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition.

There are three types of outer joins:

- LEFT OUTER
- RIGHT OUTER
- FULL OUTER

## LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
```

|     | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----|-----------|---------------|-----------------|
| 1   | Whalen    | 10            | Administration  |
| 2   | Hartstein | 20            | Marketing       |
| 3   | Fay       | 20            | Marketing       |
| 4   | Higgins   | 110           | Accounting      |
| ... |           |               |                 |
| 18  | Abel      | 80            | Sales           |
| 19  | Taylor    | 80            | Sales           |
| 20  | Grant     | (null)        | (null)          |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### Example of LEFT OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, which is the table on the left even if there is no match in the DEPARTMENTS table.

## RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
```

|     | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----|-----------|---------------|-----------------|
| 1   | Whalen    | 10            | Administration  |
| 2   | Hartstein | 20            | Marketing       |
| 3   | Fay       | 20            | Marketing       |
| ... |           |               |                 |
| 18  | Higgins   | 110           | Accounting      |
| 19  | Gietz     | 110           | Accounting      |
| 20  | (null)    | (null)        | Contracting     |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### Example of RIGHT OUTER JOIN

This query retrieves all rows in the DEPARTMENTS table, which is the table on the right even if there is no match in the EMPLOYEES table.

## FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM employees e FULL OUTER JOIN departments d
ON (e.department_id = d.department_id) ;
```

|     | LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----|-----------|---------------|-----------------|
| 1   | Whalen    | 10            | Administration  |
| 2   | Hartstein | 20            | Marketing       |
| 3   | Fay       | 20            | Marketing       |
| ... |           |               |                 |
| 18  | Abel      | 80            | Sales           |
| 19  | Taylor    | 80            | Sales           |
| 20  | Grant     | (null)        | (null)          |
| 21  | (null)    | 190           | Contracting     |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### Example of FULL OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It also retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

# Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition.



Copyright © 2009, Oracle. All rights reserved.

## Cartesian Products

When a join condition is invalid or omitted completely, the result is a *Cartesian product*, in which all combinations of rows are displayed. All rows in the first table are joined to all rows in the second table.

A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition unless you have a specific need to combine all rows from all tables.

Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

# Generating a Cartesian Product

**EMPLOYEES (20 rows)**

|            | EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|------------|-------------|-----------|---------------|
| 1          | 200         | Whalen    | 10            |
| 2          | 201         | Hartstein | 20            |
| <b>...</b> |             |           |               |
| 19         | 176         | Taylor    | 80            |
| 20         | 178         | Grant     | (null)        |

**DEPARTMENTS (8 rows)**

|   | DEPARTMENT_ID | DEPARTMENT_NAME | LOCATION_ID |
|---|---------------|-----------------|-------------|
| 1 | 10            | Administration  | 1700        |
| 2 | 20            | Marketing       | 1800        |
| 3 | 50            | Shipping        | 1500        |
| 4 | 60            | IT              | 1400        |
| 5 | 80            | Sales           | 2500        |
| 6 | 90            | Executive       | 1700        |
| 7 | 110           | Accounting      | 1700        |
| 8 | 190           | Contracting     | 1700        |

**Cartesian product:**

|   | EMPLOYEE_ID | DEPARTMENT_ID | LOCATION_ID |
|---|-------------|---------------|-------------|
| 1 | 100         | 10            | 1700        |
| 2 | 101         | 10            | 1700        |

**20 x 8 = 160 rows**

|     |     |     |      |
|-----|-----|-----|------|
| 156 | 200 | 190 | 1700 |
| 157 | 201 | 190 | 1700 |
| 158 | 202 | 190 | 1700 |
| 159 | 205 | 190 | 1700 |
| 160 | 206 | 190 | 1700 |



Copyright © 2009, Oracle. All rights reserved.

## Cartesian Products (continued)

A Cartesian product is generated if a join condition is omitted. The example in the slide shows employee last name and department name from the EMPLOYEES and DEPARTMENTS tables. Because no join condition has been specified, all rows (20 rows) from the EMPLOYEES table are joined with all rows (8 rows) in the DEPARTMENTS table, thereby generating 160 rows in the output.

## Creating Cross Joins

- The CROSS JOIN clause produces the cross-product of two tables.
- This is also called a Cartesian product between the two tables.

```
SELECT last_name, department_name
FROM employees
CROSS JOIN departments ;
```

|     | LAST_NAME | DEPARTMENT_NAME |
|-----|-----------|-----------------|
| 1   | Abel      | Administration  |
| 2   | Davies    | Administration  |
| 3   | De Haan   | Administration  |
| 4   | Ernst     | Administration  |
| ... |           |                 |
| 159 | Whalen    | Contracting     |
| 160 | Zlotkey   | Contracting     |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating Cross Joins

The example in the slide produces a Cartesian product of the EMPLOYEES and DEPARTMENTS tables.

# Summary

In this lesson, you should have learned how to use joins to display data from multiple tables by using:

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins



Copyright © 2009, Oracle. All rights reserved.

## Summary

There are multiple ways to join tables.

### Types of Joins

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins
- Cross joins
- Natural joins
- Full (or two-sided) outer joins

### Cartesian Products

A Cartesian product results in a display of all combinations of rows. This is done by either omitting the WHERE clause or specifying the CROSS JOIN clause.

### Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.

## Practice 5: Overview

This practice covers the following topics:

- Joining tables using an equijoin
- Performing outer and self-joins
- Adding conditions



Copyright © 2009, Oracle. All rights reserved.

### Practice 5: Overview

This practice is intended to give you practical experience in extracting data from multiple tables using the SQL:1999-compliant joins.

## Practice 5

1. Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output. Use a NATURAL JOIN to produce the results.

| LOCATION_ID | STREET_ADDRESS                                 | CITY                | STATE_PROVINCE | COUNTRY_NAME             |
|-------------|------------------------------------------------|---------------------|----------------|--------------------------|
| 1           | 1800 460 Bloor St. W.                          | Toronto             | Ontario        | Canada                   |
| 2           | 2500 Magdalene Centre, The Oxford Science Park | Oxford              | Oxford         | United Kingdom           |
| 3           | 1400 2014 Jabberwocky Rd                       | Southlake           | Texas          | United States of America |
| 4           | 1500 2011 Interiors Blvd                       | South San Francisco | California     | United States of America |
| 5           | 1700 2004 Charade Rd                           | Seattle             | Washington     | United States of America |

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees.

| LAST_NAME | DEPARTMENT_ID | DEPARTMENT_NAME |
|-----------|---------------|-----------------|
| Abel      | 80            | Sales           |
| Davies    | 50            | Shipping        |
| De Haan   | 90            | Executive       |
| Ernst     | 60            | IT              |
| Fay       | 20            | Marketing       |
| Gietz     | 110           | Accounting      |
| Hartstein | 20            | Marketing       |
| Higgins   | 110           | Accounting      |
| Hunold    | 60            | IT              |
| King      | 90            | Executive       |
| Kochhar   | 90            | Executive       |
| Lorentz   | 60            | IT              |
| Matos     | 50            | Shipping        |
| Mourgos   | 50            | Shipping        |
| Rajs      | 50            | Shipping        |
| Taylor    | 80            | Sales           |
| Vargas    | 50            | Shipping        |
| Whalen    | 10            | Administration  |
| Zlotkey   | 80            | Sales           |

## Practice 5 (continued)

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

| LAST_NAME   | JOB_ID | DEPARTMENT_ID | DEPARTMENT_NAME |
|-------------|--------|---------------|-----------------|
| 1 Hartstein | MK_MAN | 20            | Marketing       |
| 2 Fay       | MK_REP | 20            | Marketing       |

4. Create a report to display the last name and employee number of employees along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab\_05\_04.sql.

| Employee    | EMP#          | Manager | Mgr# |
|-------------|---------------|---------|------|
| 1 Whalen    | 200 Kochhar   | 101     |      |
| 2 Hartstein | 201 King      | 100     |      |
| 3 Fay       | 202 Hartstein | 201     |      |
| 4 Higgins   | 205 Kochhar   | 101     |      |
| 5 Gietz     | 206 Higgins   | 205     |      |
| 6 Kochhar   | 101 King      | 100     |      |
| 7 De Haan   | 102 King      | 100     |      |
| 8 Hunold    | 103 De Haan   | 102     |      |
| 9 Ernst     | 104 Hunold    | 103     |      |
| 10 Lorentz  | 107 Hunold    | 103     |      |
| 11 Mourgos  | 124 King      | 100     |      |
| 12 Rajs     | 141 Mourgos   | 124     |      |
| 13 Davies   | 142 Mourgos   | 124     |      |
| 14 Matos    | 143 Mourgos   | 124     |      |
| 15 Vargas   | 144 Mourgos   | 124     |      |
| 16 Zlotkey  | 149 King      | 100     |      |
| 17 Abel     | 174 Zlotkey   | 149     |      |
| 18 Taylor   | 176 Zlotkey   | 149     |      |
| 19 Grant    | 178 Zlotkey   | 149     |      |

## Practice 5 (continued)

5. Modify lab\_05\_04.sql to display all employees, including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab\_05\_05.sql. Run the query in lab\_05\_05.sql.

|    | Employee  | EMP# | Manager   | Mgr#   |
|----|-----------|------|-----------|--------|
| 1  | Whalen    | 200  | Kochhar   | 101    |
| 2  | Hartstein | 201  | King      | 100    |
| 3  | Fay       | 202  | Hartstein | 201    |
| 4  | Higgins   | 205  | Kochhar   | 101    |
| 5  | Gietz     | 206  | Higgins   | 205    |
| 6  | King      | 100  | (null)    | (null) |
| 7  | Kochhar   | 101  | King      | 100    |
| 8  | De Haan   | 102  | King      | 100    |
| 9  | Hunold    | 103  | De Haan   | 102    |
| 10 | Ernst     | 104  | Hunold    | 103    |

...

|    |       |     |         |     |
|----|-------|-----|---------|-----|
| 20 | Grant | 178 | Zlotkey | 149 |
|----|-------|-----|---------|-----|

6. Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named lab\_05\_06.sql.

|    | DEPARTMENT | EMPLOYEE  | COLLEAGUE |
|----|------------|-----------|-----------|
| 1  | 20         | Fay       | Hartstein |
| 2  | 20         | Hartstein | Fay       |
| 3  | 50         | Davies    | Matos     |
| 4  | 50         | Davies    | Mourgos   |
| 5  | 50         | Davies    | Rajs      |
| 6  | 50         | Davies    | Vargas    |
| 7  | 50         | Matos     | Davies    |
| 8  | 50         | Matos     | Mourgos   |
| 9  | 50         | Matos     | Rajs      |
| 10 | 50         | Matos     | Vargas    |
| 11 | 50         | Mourgos   | Davies    |
| 12 | 50         | Mourgos   | Matos     |
| 13 | 50         | Mourgos   | Rajs      |
| 14 | 50         | Mourgos   | Vargas    |

...

|    |     |         |       |
|----|-----|---------|-------|
| 42 | 110 | Higgins | Gietz |
|----|-----|---------|-------|

## Practice 5 (continued)

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB\_GRADES table, first show the structure of the JOB\_GRADES table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

| DESC JOB_GRADES |      |             |
|-----------------|------|-------------|
| Name            | Null | Type        |
| GRADE_LEVEL     |      | VARCHAR2(3) |
| LOWEST_SAL      |      | NUMBER      |
| HIGHEST_SAL     |      | NUMBER      |
| 3 rows selected |      |             |

| #  | LAST_NAME | JOB_ID     | DEPARTMENT_NAME | SALARY | GRADE_LEVEL |
|----|-----------|------------|-----------------|--------|-------------|
| 1  | Vargas    | ST_CLERK   | Shipping        | 2500   | A           |
| 2  | Matos     | ST_CLERK   | Shipping        | 2600   | A           |
| 3  | Davies    | ST_CLERK   | Shipping        | 3100   | B           |
| 4  | Rajs      | ST_CLERK   | Shipping        | 3500   | B           |
| 5  | Lorentz   | IT_PROG    | IT              | 4200   | B           |
| 6  | Whalen    | AD_ASST    | Administration  | 4400   | B           |
| 7  | Mourgos   | ST_MAN     | Shipping        | 5800   | B           |
| 8  | Ernst     | IT_PROG    | IT              | 6000   | C           |
| 9  | Fay       | MK_REP     | Marketing       | 6000   | C           |
| 10 | Gietz     | AC_ACCOUNT | Accounting      | 8300   | C           |
| 11 | Taylor    | SA_REP     | Sales           | 8600   | C           |
| 12 | Hunold    | IT_PROG    | IT              | 9000   | C           |
| 13 | Zlotkey   | SA_MAN     | Sales           | 10500  | D           |
| 14 | Abel      | SA_REP     | Sales           | 11000  | D           |
| 15 | Higgins   | AC_MGR     | Accounting      | 12000  | D           |
| 16 | Hartstein | MK_MAN     | Marketing       | 13000  | D           |
| 17 | De Haan   | AD_VP      | Executive       | 17000  | E           |
| 18 | Kochhar   | AD_VP      | Executive       | 17000  | E           |
| 19 | King      | AD_PRES    | Executive       | 24000  | E           |

## Practice 5 (continued)

If you want an extra challenge, complete the following exercises:

8. The HR department wants to determine the names of all employees who were hired after Davies.

Create a query to display the name and hire date of any employee hired after employee Davies.

|   | LAST_NAME | HIRE_DATE |
|---|-----------|-----------|
| 1 | Fay       | 17-AUG-97 |
| 2 | Lorentz   | 07-FEB-99 |
| 3 | Mourgos   | 16-NOV-99 |
| 4 | Matos     | 15-MAR-98 |
| 5 | Vargas    | 09-JUL-98 |
| 6 | Zlotkey   | 29-JAN-00 |
| 7 | Taylor    | 24-MAR-98 |
| 8 | Grant     | 24-MAY-99 |

9. The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named lab5\_09.sql.

|   | LAST_NAME | HIRE_DATE | LAST_NAME_1 | HIRE_DATE_1 |
|---|-----------|-----------|-------------|-------------|
| 1 | Whalen    | 17-SEP-87 | Kochhar     | 21-SEP-89   |
| 2 | Hunold    | 03-JAN-90 | De Haan     | 13-JAN-93   |
| 3 | Rajs      | 17-OCT-95 | Mourgos     | 16-NOV-99   |
| 4 | Davies    | 29-JAN-97 | Mourgos     | 16-NOV-99   |
| 5 | Matos     | 15-MAR-98 | Mourgos     | 16-NOV-99   |
| 6 | Vargas    | 09-JUL-98 | Mourgos     | 16-NOV-99   |
| 7 | Abel      | 11-MAY-96 | Zlotkey     | 29-JAN-00   |
| 8 | Taylor    | 24-MAR-98 | Zlotkey     | 29-JAN-00   |
| 9 | Grant     | 24-MAY-99 | Zlotkey     | 29-JAN-00   |

Oracle Internal & Oracle Academy Use Only

# Using Subqueries to Solve Queries



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

# Objectives

After completing this lesson, you should be able to do the following:

- Define subqueries
- Describe the types of problems that subqueries can solve
- List the types of subqueries
- Write single-row and multiple-row subqueries



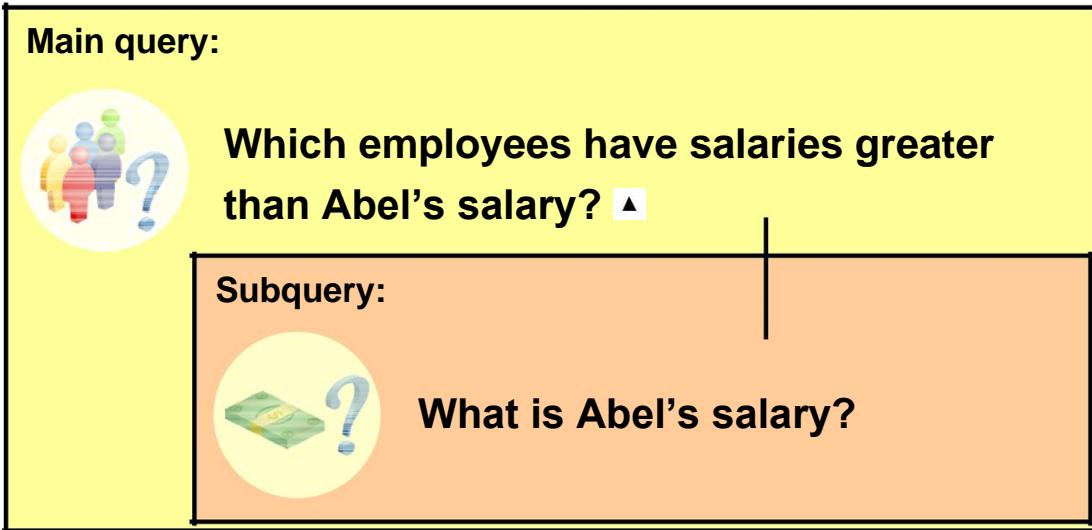
Copyright © 2009, Oracle. All rights reserved.

## Objectives

In this lesson, you learn about more advanced features of the SELECT statement. You can write subqueries in the WHERE clause of another SQL statement to obtain values based on an unknown conditional value. This lesson covers single-row subqueries and multiple-row subqueries.

# Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using a Subquery to Solve a Problem

Suppose you want to write a query to find out who earns a salary greater than Abel's salary.

To solve this problem, you need *two* queries: one to find how much Abel earns, and a second query to find who earns more than that amount.

You can solve this problem by combining the two queries, placing one query *inside* the other query.

The inner query (or *subquery*) returns a value that is used by the outer query (or *main query*). Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

# Subquery Syntax

```

SELECT select_list
FROM table
WHERE expr operator
 (SELECT select list
 FROM table) ;

```

- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query.



Copyright © 2009, Oracle. All rights reserved.

## Subquery Syntax

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including the following:

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:

*operator* includes a comparison condition such as  $>$ ,  $=$ , or  $\text{IN}$

**Note:** Comparison conditions fall into two classes: single-row operators ( $>$ ,  $=$ ,  $\geq$ ,  $<$ ,  $\neq$ ,  $\leq$ ) and multiple-row operators ( $\text{IN}$ ,  $\text{ANY}$ ,  $\text{ALL}$ ).

The subquery is often referred to as a nested SELECT, sub-SELECT, or inner SELECT statement. The subquery generally executes first, and its output is used to complete the query condition for the main (or outer) query.

# Using a Subquery

```
SELECT last_name, salary
FROM employees 11000
WHERE salary >
 (SELECT salary
 FROM employees
 WHERE last_name = 'Abel');
```

|   | LAST_NAME | SALARY |
|---|-----------|--------|
| 1 | Hartstein | 13000  |
| 2 | Higgins   | 12000  |
| 3 | King      | 24000  |
| 4 | Kochhar   | 17000  |
| 5 | De Haan   | 17000  |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Using a Subquery

In the slide, the inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.

## Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The `ORDER BY` clause in the subquery is not needed unless you are performing Top-N analysis.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.

The red horizontal bar spans most of the page width, centered below the main content area.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Guidelines for Using Subqueries

- A subquery must be enclosed in parentheses.
- Place the subquery on the right side of the comparison condition for readability.
- With Oracle8*i* and later releases, an `ORDER BY` clause can be used and is required in the subquery to perform Top-N analysis.
  - Before Oracle8*i*, however, subqueries could not contain an `ORDER BY` clause. Only one `ORDER BY` clause could be used for a `SELECT` statement; if specified, it had to be the last clause in the main `SELECT` statement.
- Two classes of comparison conditions are used in subqueries: single-row operators and multiple-row operators.

# Types of Subqueries

- Single-row subquery



- Multiple-row subquery



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Types of Subqueries

- **Single-row subqueries:** Queries that return only one row from the inner SELECT statement
- **Multiple-row subqueries:** Queries that return more than one row from the inner SELECT statement

**Note:** There are also multiple-column subqueries, which are queries that return more than one column from the inner SELECT statement. These are covered in the *Oracle Database 10g: SQL Fundamentals II* course.

# Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

| Operator | Meaning                  |
|----------|--------------------------|
| =        | Equal to                 |
| >        | Greater than             |
| >=       | Greater than or equal to |
| <        | Less than                |
| <=       | Less than or equal to    |
| <>       | Not equal to             |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Single-Row Subqueries

A single-row subquery is one that returns one row from the inner SELECT statement. This type of subquery uses a single-row operator. The slide gives a list of single-row operators.

### Example

Display the employees whose job ID is the same as that of employee 141:

```
SELECT last_name, job_id
 FROM employees
 WHERE job_id =
 (SELECT job_id
 FROM employees
 WHERE employee_id = 141);
```

| LAST_NAME | JOB_ID   |
|-----------|----------|
| Rajs      | ST_CLERK |
| Davies    | ST_CLERK |
| Matos     | ST_CLERK |
| Vargas    | ST_CLERK |

## Executing Single-Row Subqueries

```

SELECT last_name, job_id, salary
FROM employees
WHERE job_id = ST_CLERK
 (SELECT job_id
 FROM employees
 WHERE employee_id = 141)
AND salary > 2600
 (SELECT salary
 FROM employees
 WHERE employee_id = 143);

```

|   | LAST_NAME | JOB_ID   | SALARY |
|---|-----------|----------|--------|
| 1 | Rajs      | ST_CLERK | 3500   |
| 2 | Davies    | ST_CLERK | 3100   |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Executing Single-Row Subqueries

A SELECT statement can be considered as a query block. The example in the slide displays employees whose job ID is the same as that of employee 141 and whose salary is greater than that of employee 143.

The example consists of three query blocks: the outer query and two inner queries. The inner query blocks are executed first, producing the query results ST\_CLERK and 2600, respectively. The outer query block is then processed and uses the values that were returned by the inner queries to complete its search conditions.

Both inner queries return single values (ST\_CLERK and 2600, respectively), so this SQL statement is called a single-row subquery.

**Note:** The outer and inner queries can get data from different tables.

## Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM employees ← 2500
WHERE salary =
 (SELECT MIN(salary)
 FROM employees);
```

|   | LAST_NAME | JOB_ID   | SALARY |
|---|-----------|----------|--------|
| 1 | Vargas    | ST_CLERK | 2500   |



Copyright © 2009, Oracle. All rights reserved.

## Using Group Functions in a Subquery

You can display data from a main query by using a group function in a subquery to return a single row. The subquery is in parentheses and is placed after the comparison condition.

The example in the slide displays the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary. The `MIN` group function returns a single value (2500) to the outer query.

## The HAVING Clause with Subqueries

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

```

SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) > 2500
 (SELECT MIN(salary)
 FROM employees
 WHERE department_id = 50);

```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### The HAVING Clause with Subqueries

You can use subqueries not only in the WHERE clause but also in the HAVING clause. The Oracle server executes the subquery, and the results are returned into the HAVING clause of the main query. The SQL statement in the slide displays all the departments that have a minimum salary greater than that of department 50.

|   | DEPARTMENT_ID | MIN(SALARY) |
|---|---------------|-------------|
| 1 | (null)        | 7000        |
| 2 | 20            | 6000        |
| 3 | 90            | 17000       |
| 4 | 110           | 8300        |
| 5 | 80            | 8600        |
| 6 | 10            | 4400        |
| 7 | 60            | 4200        |

**Example :** Find the job with the lowest average salary.

```

SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
 FROM employees
 GROUP BY job_id);

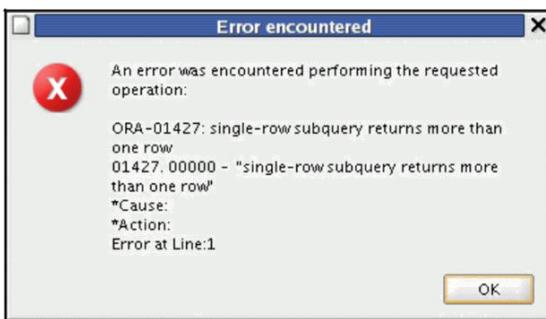
```

## What Is Wrong with This Statement?

```

SELECT employee_id, last_name
FROM employees
WHERE salary =
 (SELECT MIN(salary)
 FROM employees
 GROUP BY department_id);

```



**Single-row operator with multiple-row subquery**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Errors with Subqueries

One common error with subqueries occurs when more than one row is returned for a single-row subquery.

In the SQL statement in the slide, the subquery contains a GROUP BY clause, which implies that the subquery will return multiple rows, one for each group that it finds. In this case, the result of the subquery are 4400, 6000, 2500, 4200, 7000, 17000, and 8300.

The outer query takes those results and uses them in its WHERE clause. The WHERE clause contains an equal (=) operator, a single-row comparison operator that expects only one value. The = operator cannot accept more than one value from the subquery and, therefore, generates the error.

To correct this error, change the = operator to IN.

## Will This Statement Return Rows?

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
 (SELECT job_id
 FROM employees
 WHERE last_name = 'Haas');

0 rows selected
```

**Subquery returns no values.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Problems with Subqueries

A common problem with subqueries occurs when no rows are returned by the inner query. In the SQL statement in the slide, the subquery contains a WHERE clause. Presumably, the intention is to find the employee whose name is Haas. The statement is correct but selects no rows when executed.

There is no employee named Haas. So the subquery returns no rows. The outer query takes the results of the subquery (null) and uses these results in its WHERE clause. The outer query finds no employee with a job ID equal to null, and so returns no rows. If a job existed with a value of null, the row is not returned because comparison of two null values yields a null; therefore, the WHERE condition is not true.

## Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

| Operator | Meaning                                               |
|----------|-------------------------------------------------------|
| IN       | Equal to any member in the list                       |
| ANY      | Compare value to each value returned by the subquery  |
| ALL      | Compare value to every value returned by the subquery |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### Multiple-Row Subqueries

Subqueries that return more than one row are called multiple-row subqueries. You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values:

```
SELECT last_name, salary, department_id
 FROM employees
 WHERE salary IN (SELECT MIN(salary)
 FROM employees
 GROUP BY department_id);
```

#### Example

Find the employees who earn the same salary as the minimum salary for each department. The inner query is executed first, producing a query result. The main query block is then processed and uses the values that were returned by the inner query to complete its search condition. In fact, the main query appears to the Oracle server as follows:

```
SELECT last_name, salary,
 department_id FROM employees
 WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300,
 8600, 17000);
```

# Using the ANY Operator in Multiple-Row Subqueries

FROM  
WHERE  
AND

```
SELECT employee_id, last_name, job_id, salary
 FROM employees 9000, 6000, 4200
 WHERE salary < ANY (SELECT salary
 ...)
```

|   | EMPLOYEE_ID | LAST_NAME | JOB_ID   | SALARY |
|---|-------------|-----------|----------|--------|
| 1 | 144         | Vargas    | ST_CLERK | 2500   |
| 2 | 143         | Matos     | ST_CLERK | 2600   |

|    |     |        |            |      |
|----|-----|--------|------------|------|
| 9  | 206 | Gietz  | AC_ACCOUNT | 8500 |
| 10 | 176 | Taylor | SA_REP     | 8600 |

Copyright © 2009, Oracle. All rights reserved.

ORACLE

## Multiple-Row Subqueries (continued)

The ANY operator (and its synonym, the SOME operator) compares a value to *each* value returned by a subquery. The slide example displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is \$9,000.

<ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.

# Using the ALL Operator in Multiple-Row Subqueries

FROM  
WHERE  
AND

job\_id

```
SELECT employee_id, last_name, job_id, salary
FROM employees 9000, 6000, 4200
WHERE salary < ALL (SELECT salary
```

|   | EMPLOYEE_ID | LAST_NAME | JOB_ID   | SALARY |
|---|-------------|-----------|----------|--------|
| 1 | 141 Raji    | Rajs      | ST_CLERK | 3500   |
| 2 | 142 Davies  | Davies    | ST_CLERK | 3100   |
| 3 | 143 Matos   | Matos     | ST_CLERK | 2600   |
| 4 | 144 Vargas  | Vargas    | ST_CLERK | 2500   |

## Multiple-Row Subqueries (continued)

The ALL operator compares a value to *every* value returned by a subquery. The slide example displays employees whose salary is less than the salary of all employees with a job ID of IT\_PROG and whose job is not IT\_PROG.

>ALL means more than the maximum, and <ALL means less than the minimum.

The NOT operator can be used with IN, ANY, and ALL operators.

ORACLE

## Null Values in a Subquery

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id NOT IN
 (SELECT mgr.manager_id
 FROM employees mgr);

0 rows selected
```



Copyright © 2009, Oracle. All rights reserved.

### Returning Nulls in the Resulting Set of a Subquery

The SQL statement in the slide attempts to display all the employees who do not have any subordinates. Logically, this SQL statement should have returned 12 rows. However, the SQL statement does not return any rows. One of the values returned by the inner query is a null value, and, therefore, the entire query returns no rows.

The reason is that all conditions that compare a null value result in a null. So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator.

The NOT IN operator is equivalent to <> ALL.

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id IN
 (SELECT mgr.manager_id
 FROM employees mgr);
```

## Returning Nulls in the Resulting Set of a Subquery (continued)

Alternatively, a WHERE clause can be included in the subquery to display all employees who do not have any subordinates:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN
 (SELECT manager_id
 FROM employees
 WHERE manager_id IS NOT NULL);
```

## Summary

In this lesson, you should have learned how to:

- Identify when a subquery can help solve a question
- Write subqueries when a query is based on unknown values

```
SELECT select_list
FROM table
WHERE expr operator
 (SELECT select_list
 FROM table);
```



Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to use subqueries. A subquery is a SELECT statement that is embedded in a clause of another SQL statement. Subqueries are useful when a query is based on a search criterion with unknown intermediate values.

Subqueries have the following characteristics:

- Can pass one row of data to a main statement that contains a single-row operator, such as =, <>, >, >=, <, or <=
- Can pass multiple rows of data to a main statement that contains a multiple-row operator, such as IN
- Are processed first by the Oracle server, after which the WHERE or HAVING clause uses the results
- Can contain group functions

## Practice 6: Overview

This practice covers the following topics:

- Creating subqueries to query values based on unknown criteria
- Using subqueries to find out which values exist in one set of data and not in another



Copyright © 2009, Oracle. All rights reserved.

### Practice 6: Overview

In this practice, you write complex queries using nested SELECT statements.

#### Paper-Based Questions

You may want to create the inner query first for these questions. Make sure that it runs and produces the data that you anticipate before you code the outer query.

## Practice 6

1. The HR department needs a query that prompts users for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

| LAST_NAME | HIRE_DATE |
|-----------|-----------|
| Abel      | 11-MAY-96 |
| Taylor    | 24-MAR-98 |

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in ascending order by salary.

| EMPLOYEE_ID | LAST_NAME | SALARY |
|-------------|-----------|--------|
| 103         | Hunold    | 9000   |
| 149         | Zlotkey   | 10500  |
| 174         | Abel      | 11000  |
| 205         | Higgins   | 12000  |
| 201         | Hartstein | 13000  |
| 102         | De Haan   | 17000  |
| 101         | Kochhar   | 17000  |
| 100         | King      | 24000  |

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named lab\_06\_03.sql. Run your query.

| EMPLOYEE_ID | LAST_NAME |
|-------------|-----------|
| 124         | Mourgos   |
| 141         | Rajs      |
| 142         | Davies    |
| 143         | Matos     |
| 144         | Vargas    |
| 103         | Hunold    |
| 104         | Ernst     |
| 107         | Lorentz   |

## Practice 6 (continued)

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

| LAST_NAME | DEPARTMENT_ID | JOB_ID         |
|-----------|---------------|----------------|
| 1 Whalen  |               | 10 AD_ASST     |
| 2 Higgins |               | 110 AC_MGR     |
| 3 Gietz   |               | 110 AC_ACCOUNT |
| 4 King    |               | 90 AD_PRES     |
| 5 Kochhar |               | 90 AD_VP       |
| 6 De Haan |               | 90 AD_VP       |

Modify the query so that users are prompted for a location ID. Save this to a file named lab\_06\_04.sql.

5. Create a report for the HR department that displays the last name and salary of every employee who reports to King.

| LAST_NAME   | SALARY |
|-------------|--------|
| 1 Hartstein | 13000  |
| 2 Kochhar   | 17000  |
| 3 De Haan   | 17000  |
| 4 Mourgos   | 5800   |
| 5 Zlotkey   | 10500  |

6. Create a report for the HR department that displays the department number, last name, and job ID for every employee in the Executive department.

| DEPARTMENT_ID | LAST_NAME  | JOB_ID  |
|---------------|------------|---------|
| 1             | 90 King    | AD_PRES |
| 2             | 90 Kochhar | AD_VP   |
| 3             | 90 De Haan | AD_VP   |

If you have time, complete the following exercise:

7. Modify the query in lab\_06\_03.sql to display the employee number, last name, and salary of all the employees who earn more than the average salary and who work in a department with any employee whose last name contains a *u*. Resave lab\_06\_03.sql as lab\_06\_07.sql. Run the statement in lab\_06\_07.sql.

| EMPLOYEE_ID | LAST_NAME  | SALARY |
|-------------|------------|--------|
| 1           | 103 Hunold | 9000   |



## Using the Set Operators

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

# Objectives

After completing this lesson, you should be able to do the following:

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

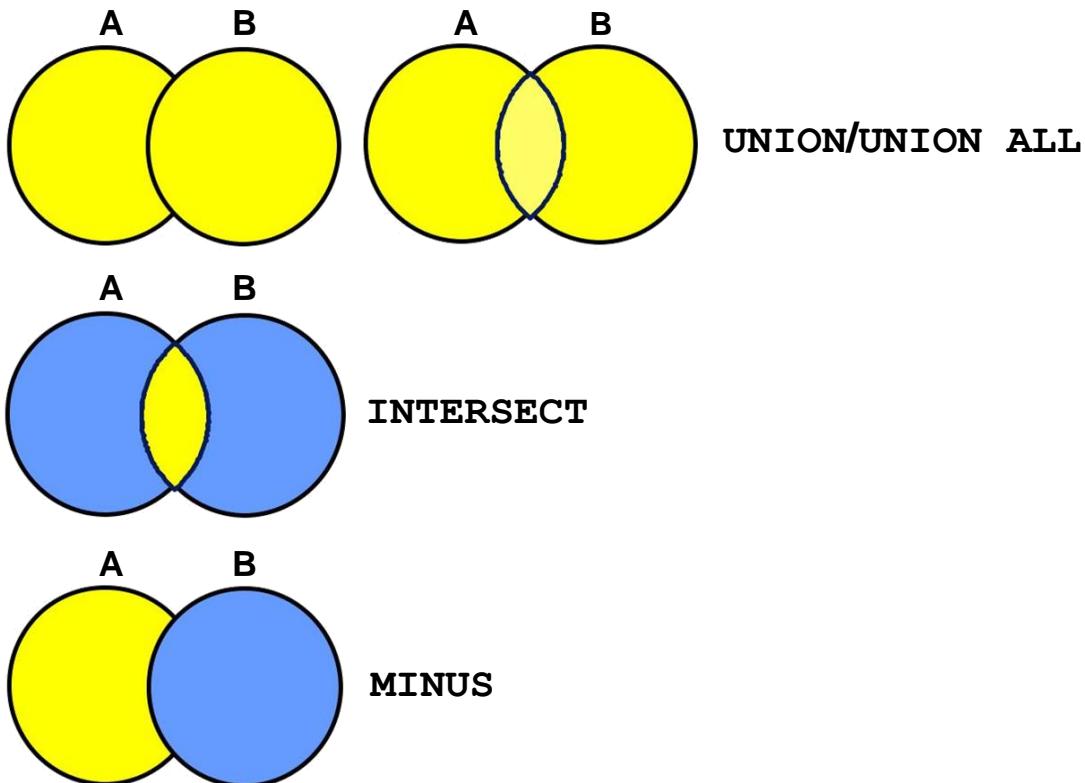


Copyright © 2009, Oracle. All rights reserved.

## Objectives

In this lesson, you learn how to write queries by using set operators.

## Set Operators



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Set Operators

Set operators combine the results of two or more component queries into one result. Queries containing set operators are called *compound queries*.

| Operator  | Returns                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------|
| UNION     | All distinct rows selected by either query                                                                        |
| UNION ALL | All rows selected by either query, including all duplicates                                                       |
| INTERSECT | All distinct rows selected by both queries                                                                        |
| MINUS     | All distinct rows that are selected by the first SELECT statement and not selected in the second SELECT statement |

All set operators have equal precedence. If a SQL statement contains multiple set operators, the Oracle server evaluates them from left (top) to right (bottom) if no parentheses explicitly specify another order. You should use parentheses to specify the order of evaluation explicitly in queries that use the INTERSECT operator with other set operators.

## Tables Used in This Lesson

The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees
- JOB\_HISTORY: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs



Copyright © 2009, Oracle. All rights reserved.

### Tables Used in This Lesson

Two tables are used in this lesson. They are the EMPLOYEES table and the JOB\_HISTORY table.

The EMPLOYEES table stores the employee details. For the human resource records, this table stores a unique identification number and e-mail address for each employee. The details of the employee's job identification number, salary, and manager are also stored. Some of the employees earn a commission in addition to their salary; this information is tracked, too. The company organizes the roles of employees into jobs. Some of the employees have been with the company for a long time and have switched to different jobs. This is monitored using the JOB\_HISTORY table. When an employee switches jobs, the details of the start date and end date of the former job, the job identification number, and the department are recorded in the JOB\_HISTORY table.

The structure and data from the EMPLOYEES and JOB\_HISTORY tables are shown on the following pages.

## Tables Used in This Lesson (continued)

There have been instances in the company of people who have held the same position more than once during their tenure with the company. For example, consider the employee Taylor, who joined the company on 24-MAR-1998. Taylor held the job title SA REP for the period 24-MAR-98 to 31-DEC-98 and the job title SA MAN for the period 01-JAN-99 to 31-DEC-99. Taylor moved back into the job title of SA REP, which is his current job title.

Similarly, consider the employee Whalen, who joined the company on 17-SEP-1987. Whalen held the job title AD ASST for the period 17-SEP-87 to 17-JUN-93 and the job title AC ACCOUNT for the period 01-JUL-94 to 31-DEC-98. Whalen moved back into the job title of AD ASST, which is his current job title.

```
DESCRIBE employees
```

| Name           | Null     | Type         |
|----------------|----------|--------------|
| EMPLOYEE_ID    | NOT NULL | NUMBER(6)    |
| FIRST_NAME     |          | VARCHAR2(20) |
| LAST_NAME      | NOT NULL | VARCHAR2(25) |
| EMAIL          | NOT NULL | VARCHAR2(25) |
| PHONE_NUMBER   |          | VARCHAR2(20) |
| HIRE_DATE      | NOT NULL | DATE         |
| JOB_ID         | NOT NULL | VARCHAR2(10) |
| SALARY         |          | NUMBER(8,2)  |
| COMMISSION_PCT |          | NUMBER(2,2)  |
| MANAGER_ID     |          | NUMBER(6)    |
| DEPARTMENT_ID  |          | NUMBER(4)    |

11 rows selected

## Tables Used in This Lesson (continued)

```
SELECT employee_id, last_name, job_id, hire_date,
department_id FROM employees;
```

| EMPLOYEE_ID | LAST_NAME     | JOB_ID     | HIRE_DATE | DEPARTMENT_ID |
|-------------|---------------|------------|-----------|---------------|
| 1           | 200 Whalen    | AD_ASST    | 17-SEP-87 | 10            |
| 2           | 201 Hartstein | MK_MAN     | 17-FEB-96 | 20            |
| 3           | 202 Fay       | MK_REP     | 17-AUG-97 | 20            |
| 4           | 205 Higgins   | AC_MGR     | 07-JUN-94 | 110           |
| 5           | 206 Gietz     | AC_ACCOUNT | 07-JUN-94 | 110           |
| 6           | 100 King      | AD_PRES    | 17-JUN-87 | 90            |
| 7           | 101 Kochhar   | AD_VP      | 21-SEP-89 | 90            |
| 8           | 102 De Haan   | AD_VP      | 13-JAN-93 | 90            |
| 9           | 103 Hunold    | IT_PROG    | 03-JAN-90 | 60            |
| 10          | 104 Ernst     | IT_PROG    | 21-MAY-91 | 60            |
| 11          | 107 Lorentz   | IT_PROG    | 07-FEB-99 | 60            |
| 12          | 124 Mourgos   | ST_MAN     | 16-NOV-99 | 50            |
| 13          | 141 Rajs      | ST_CLERK   | 17-OCT-95 | 50            |
| 14          | 142 Davies    | ST_CLERK   | 29-JAN-97 | 50            |
| 15          | 143 Matos     | ST_CLERK   | 15-MAR-98 | 50            |
| 16          | 144 Vargas    | ST_CLERK   | 09-JUL-98 | 50            |
| 17          | 149 Zlotkey   | SA_MAN     | 29-JAN-00 | 80            |
| 18          | 174 Abel      | SA_REP     | 11-MAY-96 | 80            |
| 19          | 176 Taylor    | SA_REP     | 24-MAR-98 | 80            |
| 20          | 178 Grant     | SA_REP     | 24-MAY-99 | (null)        |

```
DESCRIBE job_history
```

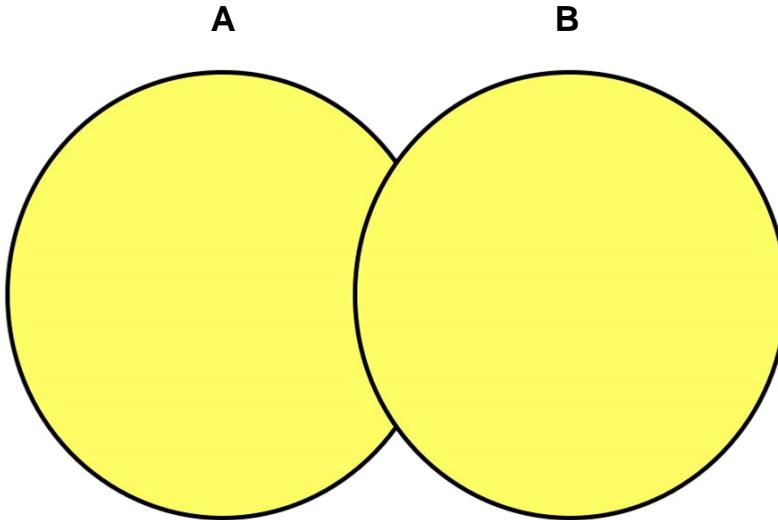
| Name            | Null     | Type         |
|-----------------|----------|--------------|
| EMPLOYEE_ID     | NOT NULL | NUMBER(6)    |
| START_DATE      | NOT NULL | DATE         |
| END_DATE        | NOT NULL | DATE         |
| JOB_ID          | NOT NULL | VARCHAR2(10) |
| DEPARTMENT_ID   |          | NUMBER(4)    |
| 5 rows selected |          |              |

## Tables Used in This Lesson (continued)

```
SELECT * FROM job_history;
```

|    | EMPLOYEE_ID | START_DATE | END_DATE  | JOB_ID     | DEPARTMENT_ID |
|----|-------------|------------|-----------|------------|---------------|
| 1  | 102         | 13-JAN-93  | 24-JUL-98 | IT_PROG    | 60            |
| 2  | 101         | 21-SEP-89  | 27-OCT-93 | AC_ACCOUNT | 110           |
| 3  | 101         | 28-OCT-93  | 15-MAR-97 | AC_MGR     | 110           |
| 4  | 201         | 17-FEB-96  | 19-DEC-99 | MK_REP     | 20            |
| 5  | 114         | 24-MAR-98  | 31-DEC-99 | ST_CLERK   | 50            |
| 6  | 122         | 01-JAN-99  | 31-DEC-99 | ST_CLERK   | 50            |
| 7  | 200         | 17-SEP-87  | 17-JUN-93 | AD_ASST    | 90            |
| 8  | 176         | 24-MAR-98  | 31-DEC-98 | SA REP     | 80            |
| 9  | 176         | 01-JAN-99  | 31-DEC-99 | SA MAN     | 80            |
| 10 | 200         | 01-JUL-94  | 31-DEC-98 | AC_ACCOUNT | 90            |

## UNION Operator



**The UNION operator returns results from both queries after eliminating duplications.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### UNION Operator

The UNION operator returns all rows that are selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.

#### Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- UNION operates over all of the columns being selected.
- NULL values are not ignored during duplicate checking.
- The IN operator has a higher precedence than the UNION operator.
- By default, the output is sorted in ascending order of the first column of the SELECT clause.

## Using the UNION Operator

Display the current and previous job details of all employees.  
Display each combination only once.

```
SELECT employee_id, job_id
FROM employees
UNION
SELECT employee_id, job_id
FROM job_history;
```

| EMPLOYEE_ID | JOB_ID         |
|-------------|----------------|
| 1           | 100 AD_PRES    |
| 2           | 101 AC_ACCOUNT |
| <b>...</b>  |                |
| 22          | 200 AC_ACCOUNT |
| 23          | 200 AD_ASST    |
| <b>...</b>  |                |
| 28          | 206 AC_ACCOUNT |


**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Using the UNION Operator

The UNION operator eliminates any duplicate records. If records that occur in both the EMPLOYEES and the JOB\_HISTORY tables are identical, the records are displayed only once. Observe in the output shown in the slide that the record for the employee with the EMPLOYEE\_ID 200 appears twice because the JOB\_ID is different in each row.

Consider the following example:

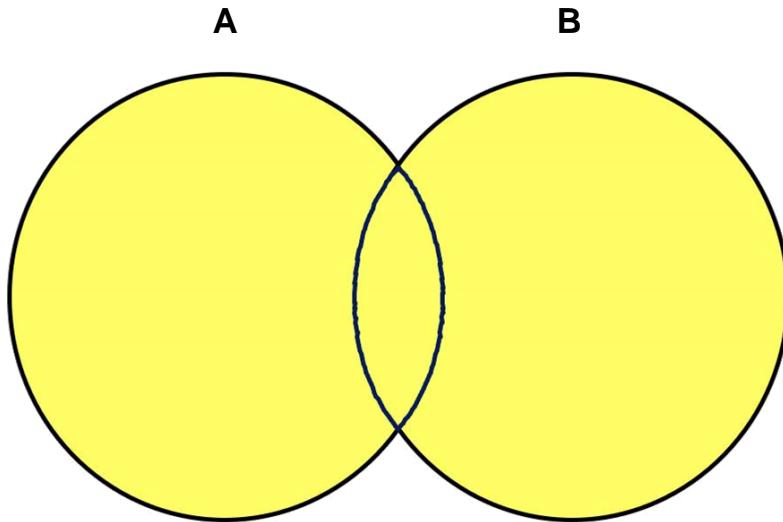
```
SELECT employee_id, job_id, department_id
FROM employees
UNION
SELECT employee_id, job_id, department_id
FROM job_history;
```

| EMPLOYEE_ID | JOB_ID         | DEPARTMENT_ID |
|-------------|----------------|---------------|
| <b>...</b>  |                |               |
| 22          | 200 AC_ACCOUNT | 90            |
| 23          | 200 AD_ASST    | 10            |
| 24          | 200 AD_ASST    | 90            |
| <b>...</b>  |                |               |

## Using the UNION Operator (continued)

In the preceding output, employee 200 appears three times. Why? Notice the DEPARTMENT\_ID values for employee 200. One row has a DEPARTMENT\_ID of 90, another 10, and the third 90. Because of these unique combinations of job IDs and department IDs, each row for employee 200 is unique and therefore not considered to be a duplicate. Observe that the output is sorted in ascending order of the first column of the SELECT clause (in this case, EMPLOYEE\_ID).

## **UNION ALL Operator**



**The UNION ALL operator returns results from both queries, including all duplications.**

**ORACLE®**

Copyright © 2009, Oracle. All rights reserved.

### **UNION ALL Operator**

Use the UNION ALL operator to return all rows from multiple queries.

#### **Guidelines**

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL:

- Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.
- The DISTINCT keyword cannot be used.

## Using the UNION ALL Operator

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

|            | EMPLOYEE_ID | JOB_ID     | DEPARTMENT_ID |
|------------|-------------|------------|---------------|
| 1          | 100         | AD_PRES    | 90            |
| 2          | 101         | AD_VP      | 90            |
| <b>...</b> |             |            |               |
| 23         | 200         | AD_ASST    | 10            |
| 24         | 200         | AC_ACCOUNT | 90            |
| 25         | 200         | AD_ASST    | 90            |
| <b>...</b> |             |            |               |
| 30         | 206         | AC_ACCOUNT | 110           |

Copyright © 2009, Oracle. All rights reserved.

### UNION ALL Operator (continued)

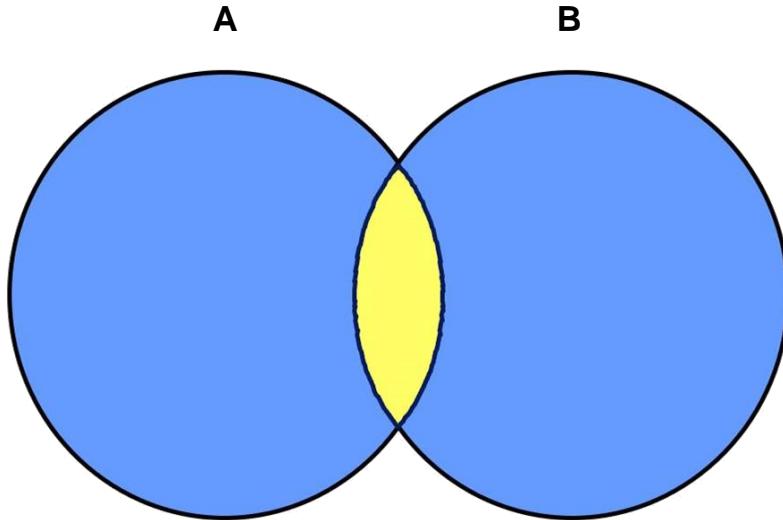
In the example, 30 rows are selected. The combination of the two tables totals to 30 rows. The UNION ALL operator does not eliminate duplicate rows. UNION returns all distinct rows selected by either query. UNION ALL returns all rows selected by either query, including all duplicates. Consider the query on the slide, now written with the UNION clause:

```
SELECT employee_id, job_id, department_id
FROM employees
UNION
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

The preceding query returns 29 rows. This is because it eliminates the following row (because it is a duplicate):

|            | EMPLOYEE_ID | JOB_ID     | DEPARTMENT_ID |
|------------|-------------|------------|---------------|
| 20         | 176         | SA_REP     | 80            |
| <b>...</b> |             |            |               |
| 29         | 206         | AC_ACCOUNT | 110           |
| <b>...</b> |             |            |               |

## **INTERSECT Operator**



**The INTERSECT operator returns rows that are common to both queries.**

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### **INTERSECT Operator**

Use the INTERSECT operator to return all rows that are common to multiple queries.

#### **Guidelines**

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- INTERSECT does not ignore NULL values.

## Using the INTERSECT Operator

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as a previous job title.

```
SELECT employee_id, job_id
FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

|   | EMPLOYEE_ID | JOB_ID  |
|---|-------------|---------|
| 1 | 176         | SA_REP  |
| 2 | 200         | AD_ASST |

Copyright © 2009, Oracle. All rights reserved.

### INTERSECT Operator (continued)

In the example in this slide, the query returns only the records that have the same values in the selected columns in both tables.

What will be the results if you add the DEPARTMENT\_ID column to the SELECT statement from the EMPLOYEES table and add the DEPARTMENT\_ID column to the SELECT statement from the JOB\_HISTORY table and run this query? The results may be different because of the introduction of another column whose values may or may not be duplicates.

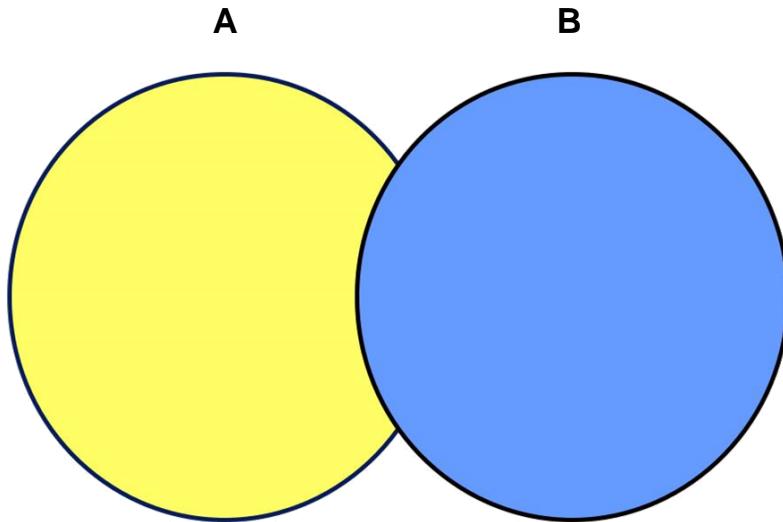
#### Example

```
SELECT employee_id, job_id, department_id
FROM employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM job_history;
```

|   | EMPLOYEE_ID | JOB_ID  |
|---|-------------|---------|
| 1 | 176         | SA_REP  |
| 2 | 200         | AD_ASST |

Employee 200 is no longer part of the results because the EMPLOYEES.DEPARTMENT\_ID value is different from the JOB\_HISTORY.DEPARTMENT\_ID value.

## MINUS Operator



**The MINUS operator returns rows in the first query  
that are not present in the second query.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

### MINUS Operator

Use the MINUS operator to return rows returned by the first query that are not present in the second query (the first SELECT statement MINUS the second SELECT statement).

**Note:** The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.

## MINUS Operator

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id
FROM employees
MINUS
SELECT employee_id
FROM job_history;
```

| EMPLOYEE_ID |
|-------------|
| 100         |
| 103         |
| 104         |
| ...         |
| 205         |
| 206         |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### MINUS Operator (continued)

In the example in the slide, the employee IDs in the JOB\_HISTORY table are subtracted from those in the EMPLOYEES table. The results set displays the employees remaining after the subtraction; they are represented by rows that exist in the EMPLOYEES table but do not exist in the JOB\_HISTORY table. These are the records of the employees who have not changed their jobs even once.

# Set Operator Guidelines

- The expressions in the SELECT lists must match in number and data type.
- Parentheses can be used to alter the sequence of execution.
- The ORDER BY clause:
  - Can appear only at the very end of the statement
  - Will accept the column name, aliases from the first SELECT statement, or the positional notation



Copyright © 2009, Oracle. All rights reserved.

## Set Operator Guidelines

- The expressions in the select lists of the queries must match in number and data type. Queries that use UNION, UNION ALL, INTERSECT, and MINUS operators in their WHERE clause must have the same number and type of columns in their SELECT list. For example:

```
SELECT employee_id, department_id
 FROM employees
 WHERE (employee_id, department_id)
 IN (SELECT employee_id, department_id
 FROM employees
 UNION
 SELECT employee_id, department_id
 FROM job_history);
```

- The ORDER BY clause:
  - Can appear only at the very end of the statement
  - Will accept the column name, an alias, or the positional notation
- The column name or alias, if used in an ORDER BY clause, must be from the first SELECT list.
- Set operators can be used in subqueries.

## Oracle Server and Set Operators

- Duplicate rows are automatically eliminated except in UNION ALL.
- Column names from the first query appear in the result.
- The output is sorted in ascending order by default except in UNION ALL.



Copyright © 2009, Oracle. All rights reserved.

### Oracle Server and Set Operators

When a query uses set operators, the Oracle server eliminates duplicate rows automatically except in the case of the UNION ALL operator. The column names in the output are decided by the column list in the first SELECT statement. By default, the output is sorted in ascending order of the first column of the SELECT clause.

The corresponding expressions in the select lists of the component queries of a compound query must match in number and data type. If component queries select character data, the data type of the return values is determined as follows:

- If both queries select values of data type CHAR, the returned values have data type CHAR.
- If either or both of the queries select values of data type VARCHAR2, the returned values have data type VARCHAR2.

## Matching the SELECT Statements

Using the UNION operator, display the department ID, location, and hire date for all employees.

```
SELECT department_id, TO_NUMBER(null)
 location, hire_date
 FROM employees
UNION
SELECT department_id, location_id, TO_DATE(null)
 FROM departments;
```

|     | DEPARTMENT_ID | LOCATION         | HIRE_DATE |
|-----|---------------|------------------|-----------|
| 1   | 10            | 1700 (null)      |           |
| 2   | 10            | (null) 17-SEP-87 |           |
| 3   | 20            | 1800 (null)      |           |
| ... |               |                  |           |
| 26  | 190           | 1700 (null)      |           |
| 27  | (null)        | (null) 24-MAY-99 |           |



Copyright © 2009, Oracle. All rights reserved.

## Matching the SELECT Statements

Because the expressions in the select lists of the queries must match in number, you can use dummy columns and the data type conversion functions to comply with this rule. In the slide, the name `location` is given as the dummy column heading. The `TO_NUMBER` function is used in the first query to match the `NUMBER` data type of the `LOCATION_ID` column retrieved by the second query. Similarly, the `TO_DATE` function in the second query is used to match the `DATE` data type of the `HIRE_DATE` column retrieved by the first query.

## Matching the SELECT Statement: Example

Using the UNION operator, display the employee ID, job ID, and salary of all employees.

```
SELECT employee_id, job_id, salary
FROM employees
UNION
SELECT employee_id, job_id, 0
FROM job_history;
```

|     | EMPLOYEE_ID | JOB_ID     | SALARY |
|-----|-------------|------------|--------|
| 1   | 100         | AD_PRES    | 24000  |
| 2   | 101         | AC_ACCOUNT | 0      |
| 3   | 101         | AC_MGR     | 0      |
| ... |             |            |        |
| 29  | 205         | AC_MGR     | 12000  |
| 30  | 206         | AC_ACCOUNT | 8300   |

Copyright © 2009, Oracle. All rights reserved.

### Matching the SELECT Statement: Example

The EMPLOYEES and JOB\_HISTORY tables have several columns in common (for example, EMPLOYEE\_ID, JOB\_ID, and DEPARTMENT\_ID). But what if you want the query to display the employee ID, job ID, and salary using the UNION operator, knowing that the salary exists only in the EMPLOYEES table?

The code example in the slide matches the EMPLOYEE\_ID and JOB\_ID columns in the EMPLOYEES and JOB\_HISTORY tables. A literal value of 0 is added to the JOB\_HISTORY SELECT statement to match the numeric SALARY column in the EMPLOYEES SELECT statement.

In the preceding results, each row in the output that corresponds to a record from the JOB\_HISTORY table contains a 0 in the SALARY column.

## Controlling the Order of Rows

Produce an English sentence using two UNION operators.

```
COLUMN a_dummy NOPRINT
SELECT 'sing' AS "My dream", 3 a_dummy
FROM dual
UNION
SELECT 'I''d like to teach', 1 a_dummy
FROM dual
UNION
SELECT 'the world to', 2 a_dummy
FROM dual
ORDER BY a_dummy;
```

|   | My dream          | A_DUMMY |
|---|-------------------|---------|
| 1 | I'd like to teach | 1       |
| 2 | the world to      | 2       |
| 3 | sing              | 3       |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Controlling the Order of Rows

By default, the output is sorted in ascending order on the first column. You can use the ORDER BY clause to change this.

The ORDER BY clause can be used only once in a compound query. If used, the ORDER BY clause must be placed at the end of the query. The ORDER BY clause accepts the column name or an alias. Without the ORDER BY clause, the code example in the slide produces the following output in the alphabetical order of the first column:

|   | My dream          | A_DUMMY |
|---|-------------------|---------|
| 1 | I'd like to teach | 1       |
| 2 | sing              | 3       |
| 3 | the world to      | 2       |

**Note:** Consider a compound query where the UNION set operator is used more than once. In this case, the ORDER BY clause can use only positions rather than explicit expressions.

## Summary

In this lesson, you should have learned how to use:

- UNION to return all distinct rows
- UNION ALL to return all rows, including duplicates
- INTERSECT to return all rows that are shared by both queries
- MINUS to return all distinct rows that are selected by the first query but not by the second
- ORDER BY only at the very end of the statement



Copyright © 2009, Oracle. All rights reserved.

## Summary

- The UNION operator returns all rows selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.
- Use the UNION ALL operator to return all rows from multiple queries. Unlike the case with the UNION operator, duplicate rows are not eliminated and the output is not sorted by default.
- Use the INTERSECT operator to return all rows that are common to multiple queries.
- Use the MINUS operator to return rows returned by the first query that are not present in the second query.
- Remember to use the ORDER BY clause only at the very end of the compound statement.
- Make sure that the corresponding expressions in the SELECT lists match in number and data type.

## Practice 7: Overview

In this practice, you use the following set operators to create reports:

- UNION operator
- INTERSECTION operator
- MINUS operator



Copyright © 2009, Oracle. All rights reserved.

### Practice 7: Overview

In this practice, you write queries using the set operators.

## Practice 7

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST\_CLERK. Use set operators to create this report.

| DEPARTMENT_ID |
|---------------|
| 10            |
| 20            |
| 60            |
| 80            |
| 90            |
| 110           |
| 190           |

2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use set operators to create this report.

| COUNTRY_ID | COUNTRY_NAME |
|------------|--------------|
| DE         | Germany      |

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display the job ID and department ID using set operators.

| JOB_ID   | DEPARTMENT_ID | DUMMY |
|----------|---------------|-------|
| AD_ASST  | 10            | x     |
| ST_CLERK | 50            | y     |
| ST_MAN   | 50            | y     |
| MK_MAN   | 20            | z     |
| MK_REP   | 20            | z     |

4. Create a report that lists the employee ID and job ID of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).

| EMPLOYEE_ID | JOB_ID  |
|-------------|---------|
| 176         | SA_REP  |
| 200         | AD_ASST |

## Practice 7 (continued)

5. The HR department needs a report with the following specifications:

- The last name and department ID of all employees from the EMPLOYEES table, regardless of whether or not they belong to a department
- The department ID and department name of all departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them

Write a compound query to accomplish this.

|    | LAST_NAME | DEPARTMENT_ID | TO_CHAR(NULL)     |
|----|-----------|---------------|-------------------|
| 1  | Abel      |               | 80 (null)         |
| 2  | Davies    |               | 50 (null)         |
| 3  | De Haan   |               | 90 (null)         |
| 4  | Ernst     |               | 60 (null)         |
| 5  | Fay       |               | 20 (null)         |
| 6  | Gietz     |               | 110 (null)        |
| 7  | Grant     |               | (null) (null)     |
| 8  | Hartstein |               | 20 (null)         |
| 9  | Higgins   |               | 110 (null)        |
| 10 | Hunold    |               | 60 (null)         |
| 11 | King      |               | 90 (null)         |
| 12 | Kochhar   |               | 90 (null)         |
| 13 | Lorentz   |               | 60 (null)         |
| 14 | Matos     |               | 50 (null)         |
| 15 | Mourgos   |               | 50 (null)         |
| 16 | Rajs      |               | 50 (null)         |
| 17 | Taylor    |               | 80 (null)         |
| 18 | Vargas    |               | 50 (null)         |
| 19 | Whalen    |               | 10 (null)         |
| 20 | Zlotkey   |               | 80 (null)         |
| 21 | (null)    |               | 10 Administration |
| 22 | (null)    |               | 20 Marketing      |
| 23 | (null)    |               | 50 Shipping       |
| 24 | (null)    |               | 60 IT             |
| 25 | (null)    |               | 80 Sales          |
| 26 | (null)    |               | 90 Executive      |
| 27 | (null)    |               | 110 Accounting    |
| 28 | (null)    |               | 190 Contracting   |

Oracle Internal & Oracle Academy Use Only

# 8

## Manipulating Data

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

# Objectives

After completing this lesson, you should be able to do the following:

- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Control transactions



Copyright © 2009, Oracle. All rights reserved.

## Objective

In this lesson, you learn how to use DML statements to insert rows into a table, update existing rows in a table, and delete existing rows from a table. You also learn how to control transactions with the COMMIT, SAVEPOINT, and ROLLBACK statements.

# Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

The red bar spans most of the page width, centered horizontally.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Data Manipulation Language

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction*.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account, and record the transaction in the transaction journal. The Oracle server must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

## Adding a New Row to a Table

**DEPARTMENTS**

New row

Insert new row into the DEPARTMENTS table

|   | DEPARTMENT_ID     | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|-------------------|-----------------|------------|-------------|
| 1 | 10 Administration |                 | 200        | 1700        |
| 2 | 20 Marketing      |                 | 201        | 1800        |
| 3 | 50 Shipping       |                 | 124        | 1500        |
| 4 | 60 IT             |                 | 103        | 1400        |
| 5 | 80 Sales          |                 | 149        | 2500        |
| 6 | 90 Executive      |                 | 100        | 1700        |
| 7 | 110 Accounting    |                 | 205        | 1700        |
| 8 | 190 Contracting   | (null)          |            | 1700        |

|   | DEPARTMENT_ID     | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|-------------------|-----------------|------------|-------------|
| 1 | 10 Administration |                 | 200        | 1700        |
| 2 | 20 Marketing      |                 | 201        | 1800        |
| 3 | 50 Shipping       |                 | 124        | 1500        |
| 4 | 60 IT             |                 | 103        | 1400        |
| 5 | 80 Sales          |                 | 149        | 2500        |
| 6 | 90 Executive      |                 | 100        | 1700        |
| 7 | 110 Accounting    |                 | 205        | 1700        |
| 8 | 190 Contracting   | (null)          |            | 1700        |
| 9 | 190 Contracting   | (null)          |            | 1700        |

Copyright © 2009, Oracle. All rights reserved.

ORACLE

## Adding a New Row to a Table

The graphic in the slide illustrates adding a new department to the DEPARTMENTS table.

## INSERT Statement Syntax

- Add new rows to a table by using the `INSERT` statement:

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

- With this syntax, only one row is inserted at a time.



Copyright © 2009, Oracle. All rights reserved.

### Adding a New Row to a Table (continued)

You can add new rows to a table by issuing the `INSERT` statement.

In the syntax:

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>table</i>  | is the name of the table                           |
| <i>column</i> | is the name of the column in the table to populate |
| <i>value</i>  | is the corresponding value for the column          |

**Note:** This statement with the `VALUES` clause adds only one row at a time to a table.

## Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
INSERT INTO departments(department_id,
 department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100,
1700); 1 rows inserted
```

- Enclose character and date values in single quotation marks.



Copyright © 2009, Oracle. All rights reserved.

### Adding a New Row to a Table (continued)

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

| DESCRIBE departments |          |              |
|----------------------|----------|--------------|
| Name                 | Null     | Type         |
| DEPARTMENT_ID        | NOT NULL | NUMBER(4)    |
| DEPARTMENT_NAME      | NOT NULL | VARCHAR2(30) |
| MANAGER_ID           |          | NUMBER(6)    |
| LOCATION_ID          |          | NUMBER(4)    |
| 4 rows selected      |          |              |

For clarity, use the column list in the `INSERT` clause.

Enclose character and date values in single quotation marks; it is not recommended that you enclose numeric values in single quotation marks.

Number values should not be enclosed in single quotation marks, because implicit conversion may take place for numeric values that are assigned to `NUMBER` data type columns if single quotation marks are included.

## Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO departments (department_id,
 department_name)
VALUES (30, 'Purchasing');
1 rows inserted
```

- Explicit method: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO departments
VALUES (100, 'Finance', NULL, NULL);
rows inserted
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Methods for Inserting Null Values

| Method   | Description                                                                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------------------|
| Implicit | Omit the column from the column list.                                                                                                 |
| Explicit | Specify the NULL keyword in the VALUES list;<br>specify the empty string (' ') in the VALUES list for character strings<br>and dates. |

Be sure that you can use null values in the targeted column by verifying the Null? status with the DESCRIBE command.

The Oracle server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input:

- Mandatory value missing for a NOT NULL column
- Duplicate value violates uniqueness constraint
- Foreign key constraint violated
- CHECK constraint violated
- Data type mismatch
- Value too wide to fit in column

# Inserting Special Values

The SYSDATE function records the current date and time.

```
INSERT INTO employees (employee_id,
 first_name, last_name,
 email, phone_number,
 hire_date, job_id, salary,
 commission_pct, manager_id,
 department_id)
VALUES
 (113,
 'Louis', 'Popp',
 'LPOPP', '515.124.4567',
 SYSDATE, 'AC_ACCOUNT', 6900,
 NULL, 205, 100);
1 rows inserted
```

Copyright © 2009, Oracle. All rights reserved.

## Inserting Special Values by Using SQL Functions

You can use functions to enter special values in your table.

The slide example records information for employee Popp in the EMPLOYEES table. It supplies the current date and time in the HIRE\_DATE column. It uses the SYSDATE function for current date and time.

You can also use the USER function when inserting rows in a table. The USER function records the current username.

### Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date,
commission_pct FROM employees
WHERE employee_id = 113;
```

|   | EMPLOYEE_ID | LAST_NAME | JOB_ID     | HIRE_DATE | COMMISSION_PCT |
|---|-------------|-----------|------------|-----------|----------------|
| 1 | 113         | Popp      | AC_ACCOUNT | 10-NOV-08 | (null)         |

## Inserting Specific Date Values

- Add a new employee.

```
INSERT INTO employees
VALUES (114,
 'Den', 'Raphealy',
 'DRAPHEAL', '515.127.4561',
 TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
 'AC_ACCOUNT', 11000, NULL, 100, 30);
1 rows inserted
```

- Verify your addition.

|   | EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL    | PHONE_NUMBER | HIRE_DATE | JOB_ID     | SALARY | COMMISSION_PCT |
|---|-------------|------------|-----------|----------|--------------|-----------|------------|--------|----------------|
| 1 | 114         | Den        | Raphealy  | DRAPHEAL | 515.127.4561 | 03-FEB-99 | AC_ACCOUNT | 11000  | (null)         |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Inserting Specific Date and Time Values

The DD-MON-YY format is usually used to insert a date value. With this format, recall that the century defaults to the current century. Because the date also contains time information, the default time is midnight (00:00:00).

If a date must be entered in a format other than the default format (for example, with another century or a specific time), you must use the TO\_DATE function.

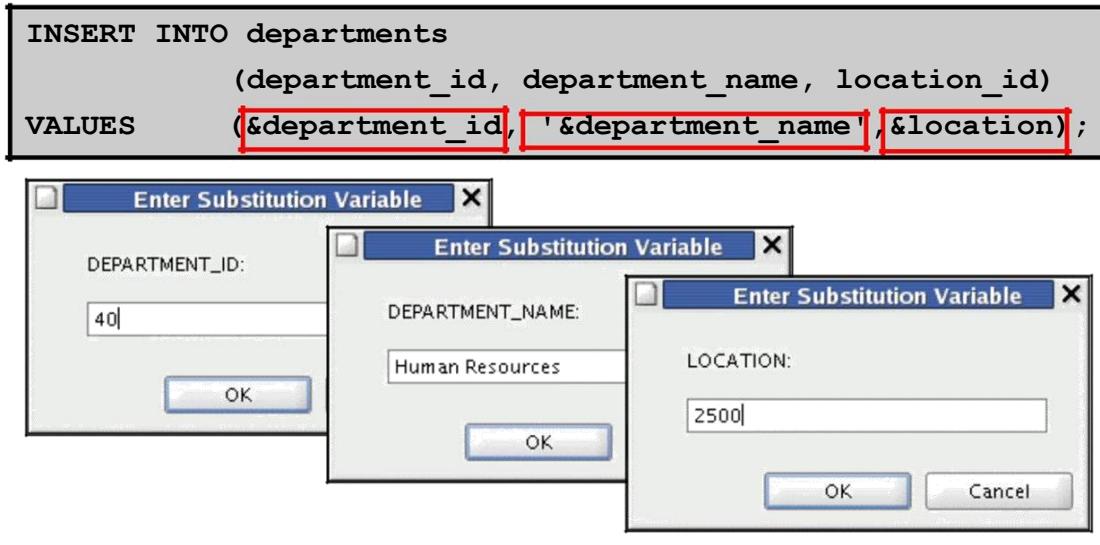
The example in the slide records information for employee Raphealy in the EMPLOYEES table. It sets the HIRE\_DATE column to be February 3, 1999. If you use the following statement instead of the one shown in the slide, the year of the hire date is interpreted as 2099.

```
INSERT INTO employees
VALUES (114,
 'Den', 'Raphealy',
 'DRAPHEAL', '515.127.4561',
 '03-FEB-99',
 'AC_ACCOUNT', 11000, NULL, 100, 30);
```

If the RR format is used, the system provides the correct century automatically, even if it is not the current one.

## Creating a Script

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.



Copyright © 2009, Oracle. All rights reserved.

### Creating a Script to Manipulate Data

You can save commands with substitution variables to a file and execute the commands in the file. The slide example records information for a department in the DEPARTMENTS table.

Run the script file and you are prompted for input for each of the & substitution variables. After entering a value for the substitution variable, click the Continue button. The values that you input are then substituted into the statement. This enables you to run the same script file over and over but supply a different set of values each time you run it.

## Copying Rows from Another Table

- Write your `INSERT` statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

4 rows inserted

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.



Copyright © 2009, Oracle. All rights reserved.

## Copying Rows from Another Table

You can use the `INSERT` statement to add rows to a table where the values are derived from existing tables. In place of the `VALUES` clause, you use a subquery. In the slide example, for the `INSERT INTO` statement to work, you must have already created the `sales_reps` table using the `CREATE TABLE` statement. `CREATE TABLE` is discussed in the next lesson titled “Using DDL Statements to Create and Manage Tables.”

### Syntax

```
INSERT INTO table [column (, column)] subquery;
```

In the syntax:

`table` is the table name  
`column` is the name of the column in the table to populate  
`subquery` is the subquery that returns rows to the table

The number of columns and their data types in the column list of the `INSERT` clause must match the number of values and their data types in the subquery. To create a copy of the rows of a table, use `SELECT *` in the subquery:

```
INSERT INTO copy_emp
SELECT *
FROM employees;
```

# Changing Data in a Table

## EMPLOYEES

| EMPLOYEE_ID   | FIRST_NAME | LAST_NAME | EMAIL     | HIRE_DATE | JOB_ID | SALARY | DEPARTMENT_ID | COMMISSION_PCT |
|---------------|------------|-----------|-----------|-----------|--------|--------|---------------|----------------|
| 100 Steven    | King       | SKING     | 17-JUN-87 | AD_PRES   | 24000  | 90     | (null)        |                |
| 101 Neena     | Kochhar    | NKOCHHAR  | 21-SEP-89 | AD_VP     | 17000  | 90     | (null)        |                |
| 102 Lex       | De Haan    | LDEHAAN   | 13-JAN-93 | AD_VP     | 17000  | 90     | (null)        |                |
| 103 Alexander | Hunold     | AHUNOLD   | 03-JAN-90 | IT_PROG   | 9000   | 60     | (null)        |                |
| 104 Bruce     | Ernst      | BERNST    | 21-MAY-91 | IT_PROG   | 6000   | 60     | (null)        |                |
| 107 Diana     | Lorentz    | DLORENTZ  | 07-FEB-99 | IT_PROG   | 4200   | 60     | (null)        |                |
| 124 Kevin     | Mourgos    | KMOURGOS  | 16-NOV-99 | ST_MAN    | 5800   | 50     | (null)        |                |

Update rows in the EMPLOYEES table:



| EMPLOYEE_ID   | FIRST_NAME | LAST_NAME | EMAIL     | HIRE_DATE | JOB_ID | SALARY | DEPARTMENT_ID | COMMISSION_PCT |
|---------------|------------|-----------|-----------|-----------|--------|--------|---------------|----------------|
| 100 Steven    | King       | SKING     | 17-JUN-87 | AD_PRES   | 24000  | 90     | (null)        |                |
| 101 Neena     | Kochhar    | NKOCHHAR  | 21-SEP-89 | AD_VP     | 17000  | 90     | (null)        |                |
| 102 Lex       | De Haan    | LDEHAAN   | 13-JAN-93 | AD_VP     | 17000  | 90     | (null)        |                |
| 103 Alexander | Hunold     | AHUNOLD   | 03-JAN-90 | IT_PROG   | 9000   | 30     | (null)        |                |
| 104 Bruce     | Ernst      | BERNST    | 21-MAY-91 | IT_PROG   | 6000   | 30     | (null)        |                |
| 107 Diana     | Lorentz    | DLORENTZ  | 07-FEB-99 | IT_PROG   | 4200   | 30     | (null)        |                |
| 124 Kevin     | Mourgos    | KMOURGOS  | 16-NOV-99 | ST_MAN    | 5800   | 50     | (null)        |                |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Changing Data in a Table

The slide illustrates changing the department number for employees in department 60 to department 30.

## UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement:

```
UPDATE table
SET column = value [, column = value, ...]
[WHERE condition];
```

- Update more than one row at a time (if required).



Copyright © 2009, Oracle. All rights reserved.

### Updating Rows

You can modify existing rows by using the UPDATE statement.

In the syntax:

|                  |                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>table</i>     | is the name of the table                                                                                                        |
| <i>column</i>    | is the name of the column in the table to populate                                                                              |
| <i>value</i>     | is the corresponding value or subquery for the column                                                                           |
| <i>condition</i> | identifies the rows to be updated and is composed of column names, expressions, constants, subqueries, and comparison operators |

Confirm the update operation by querying the table to display the updated rows.

For more information, see “UPDATE” in the *Oracle Database SQL Reference*.

**Note:** In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

## Updating Rows in a Table

- Specific row or rows are modified if you specify the WHERE clause:

```
UPDATE employees
SET department_id = 70
WHERE employee_id = 113;
1 rows updated
```

- All rows in the table are modified if you omit the WHERE clause:

```
UPDATE copy_emp
SET department_id = 110;
22 rows updated
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Updating Rows (continued)

The UPDATE statement modifies specific rows if the WHERE clause is specified. The slide example transfers employee 113 (Popp) to department 70.

If you omit the WHERE clause, all the rows in the table are modified.

```
SELECT last_name, department_id
FROM copy_emp;
```

|     | LAST_NAME | DEPARTMENT_ID |
|-----|-----------|---------------|
| 1   | Whalen    | 110           |
| 2   | Hartstein | 110           |
| 3   | Fay       | 110           |
| 4   | Higgins   | 110           |
| 5   | Gietz     | 110           |
| 6   | Raphealy  | 110           |
| ... |           |               |

**Note:** The COPY\_EMP table has the same data as the EMPLOYEES table.

## Updating Two Columns with a Subquery

Update employee 114's job and salary to match that of employee 205.

```
UPDATE employees
SET job_id = (SELECT job_id
 FROM employees
 WHERE employee_id = 205) ,
 salary = (SELECT salary
 FROM employees
 WHERE employee_id = 205)
WHERE employee_id = 114;
1 rows updated
```

Copyright © 2009, Oracle. All rights reserved.

## Updating Two Columns with a Subquery

You can update multiple columns in the SET clause of an UPDATE statement by writing multiple subqueries.

### Syntax

```
UPDATE table
SET column =
 (SELECT column
 FROM table
 WHERE condition)
[, column =
 (SELECT column
 FROM table
 WHERE condition)]
[WHERE condition] ;
```

**Note:** If no rows are updated, the message “0 rows updated” is returned.

## Updating Rows Based on Another Table

Use subqueries in UPDATE statements to update rows in a table based on values from another table:

```
UPDATE copy.emp
SET department_id = (SELECT department_id
 FROM employees
 WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
 FROM employees
 WHERE employee_id = 200);
1 rows updated
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Updating Rows Based on Another Table

You can use subqueries in UPDATE statements to update rows in a table. The example in the slide updates the COPY\_EMP table based on the values from the EMPLOYEES table. It changes the department number of all employees with employee 200's job ID to employee 100's current department number.

# Removing a Row from a Table

## DEPARTMENTS

|   | DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|---------------|-----------------|------------|-------------|
| 1 | 30            | Purchasing      | (null)     | (null)      |
| 2 | 40            | Human Resources | (null)     | 2500        |
| 3 | 10            | Administration  | 200        | 1700        |
| 4 | 20            | Marketing       | 201        | 1800        |
| 5 | 50            | Shipping        | 124        | 1500        |
| 6 | 60            | IT              | 103        | 1400        |

Delete a row from the DEPARTMENTS table:

|   | DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|---------------|-----------------|------------|-------------|
| 1 | 40            | Human Resources | (null)     | 2500        |
| 2 | 10            | Administration  | 200        | 1700        |
| 3 | 20            | Marketing       | 201        | 1800        |
| 4 | 50            | Shipping        | 124        | 1500        |
| 5 | 60            | IT              | 103        | 1400        |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Removing a Row from a Table

The graphic in the slide removes the Purchasing department from the DEPARTMENTS table (assuming that there are no constraints defined on the DEPARTMENTS table).

## DELETE Statement

You can remove existing rows from a table by using the `DELETE` statement:

```
DELETE [FROM] table
[WHERE] condition ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Deleting Rows

You can remove existing rows by using the `DELETE` statement.

In the syntax:

*table* is the table name

*condition* identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries, and comparison operators

**Note:** If no rows are deleted, the message “0 rows deleted” is returned.

For more information, see “`DELETE`” in the *Oracle Database SQL Reference*.

## Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 rows deleted
```

- All rows in the table are deleted if you omit the WHERE clause:

```
DELETE FROM copy_emp;
22 rows deleted
```



Copyright © 2009, Oracle. All rights reserved.

### Deleting Rows (continued)

You can delete specific rows by specifying the WHERE clause in the DELETE statement. The slide example deletes the Finance department from the DEPARTMENTS table. You can confirm the delete operation by displaying the deleted rows using the SELECT statement.

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
0 rows selected.
```

If you omit the WHERE clause, all rows in the table are deleted. The second example in the slide deletes all the rows from the COPY\_EMP table, because no WHERE clause has been specified.

#### Example

Remove rows identified in the WHERE clause.

```
DELETE FROM employees WHERE employee_id = 114;
1 rows deleted.
```

```
DELETE FROM departments WHERE department_id IN (30, 40);
2 rows deleted.
```

## Deleting Rows Based on Another Table

Use subqueries in DELETE statements to remove rows from a table based on values from another table:

```
DELETE FROM employees
WHERE department_id =
 (SELECT department_id
 FROM departments
 WHERE department_name
 LIKE '%Public%');

1 rows deleted
```



Copyright © 2009, Oracle. All rights reserved.

### Deleting Rows Based on Another Table

You can use subqueries to delete rows from a table based on values from another table. The example in the slide deletes all the employees who are in a department where the department name contains the string Public. The subquery searches the DEPARTMENTS table to find the department number based on the department name containing the string Public. The subquery then feeds the department number to the main query, which deletes rows of data from the EMPLOYEES table based on this department number.

## TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```



Copyright © 2009, Oracle. All rights reserved.

## TRUNCATE Statement

A more efficient method of emptying a table is with the TRUNCATE statement.

You can use the TRUNCATE statement to quickly remove all rows from a table or cluster.

Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information. Rollback information is covered later in this lesson.
- Truncating a table does not fire the delete triggers of the table.
- If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement. Disabling constraints is covered in a subsequent lesson.

## Using a Subquery in an INSERT Statement

```
INSERT INTO
 (SELECT employee_id, last_name,
 email, hire_date, job_id, salary,
 department_id
 FROM employees
 WHERE department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
 TO_DATE('07-JUN-99', 'DD-MON-RR'),
 'ST_CLERK', 5000, 50);

1 rows inserted
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Using a Subquery in an INSERT Statement

You can use a subquery in place of the table name in the `INTO` clause of the `INSERT` statement. The select list of this subquery must have the same number of columns as the column list of the `VALUES` clause. Any rules on the columns of the base table must be followed if the `INSERT` statement is to work successfully. For example, you could not put in a duplicate employee ID or omit a value for a mandatory not-null column.

# Using a Subquery in an INSERT Statement

Verify the results:

```
SELECT employee_id, last_name, email,
 hire_date, job_id, salary, department_id
 FROM employees
 WHERE department_id = 50;
```

|   | EMPLOYEE_ID | LAST_NAME | EMAIL    | HIRE_DATE | JOB_ID   | SALARY | DEPARTMENT_ID |
|---|-------------|-----------|----------|-----------|----------|--------|---------------|
| 1 | 99999       | Taylor    | DTAYLOR  | 07-JUN-99 | ST_CLERK | 5000   | 50            |
| 2 | 124         | Mourgos   | KMOURGOS | 16-NOV-99 | ST_MAN   | 5800   | 50            |
| 3 | 141         | Rajs      | TRAJS    | 17-OCT-95 | ST_CLERK | 3500   | 50            |
| 4 | 142         | Davies    | CDAVIES  | 29-JAN-97 | ST_CLERK | 3100   | 50            |
| 5 | 143         | Matos     | RMATOS   | 15-MAR-98 | ST_CLERK | 2600   | 50            |
| 6 | 144         | Vargas    | PVARGAS  | 09-JUL-98 | ST_CLERK | 2500   | 50            |

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

## Using a Subquery in an INSERT Statement (continued)

The example shows the results of the subquery that was used to identify the table for the INSERT statement.

# Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement



Copyright © 2009, Oracle. All rights reserved.

## Database Transactions

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that make up one consistent change to the data. For example, a transfer of funds between two accounts should include the debit to one account and the credit to another account in the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

## Transaction Types

| Type                             | Description                                                                                                         |
|----------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Data manipulation language (DML) | Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work |
| Data definition language (DDL)   | Consists of only one DDL statement                                                                                  |
| Data control language (DCL)      | Consists of only one DCL statement                                                                                  |

# Database Transactions

- Begin when the first DML SQL statement is executed
- End with one of the following events:
  - A COMMIT or ROLLBACK statement is issued.
  - A DDL or DCL statement executes (automatic commit).
  - The user exits SQL Developer or SQL\*Plus.
  - The system crashes.



Copyright © 2009, Oracle. All rights reserved.

## When Does a Transaction Start and End?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A COMMIT or ROLLBACK statement is issued.
- A DDL statement, such as CREATE, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL\*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

## **Advantages of COMMIT and ROLLBACK Statements**

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

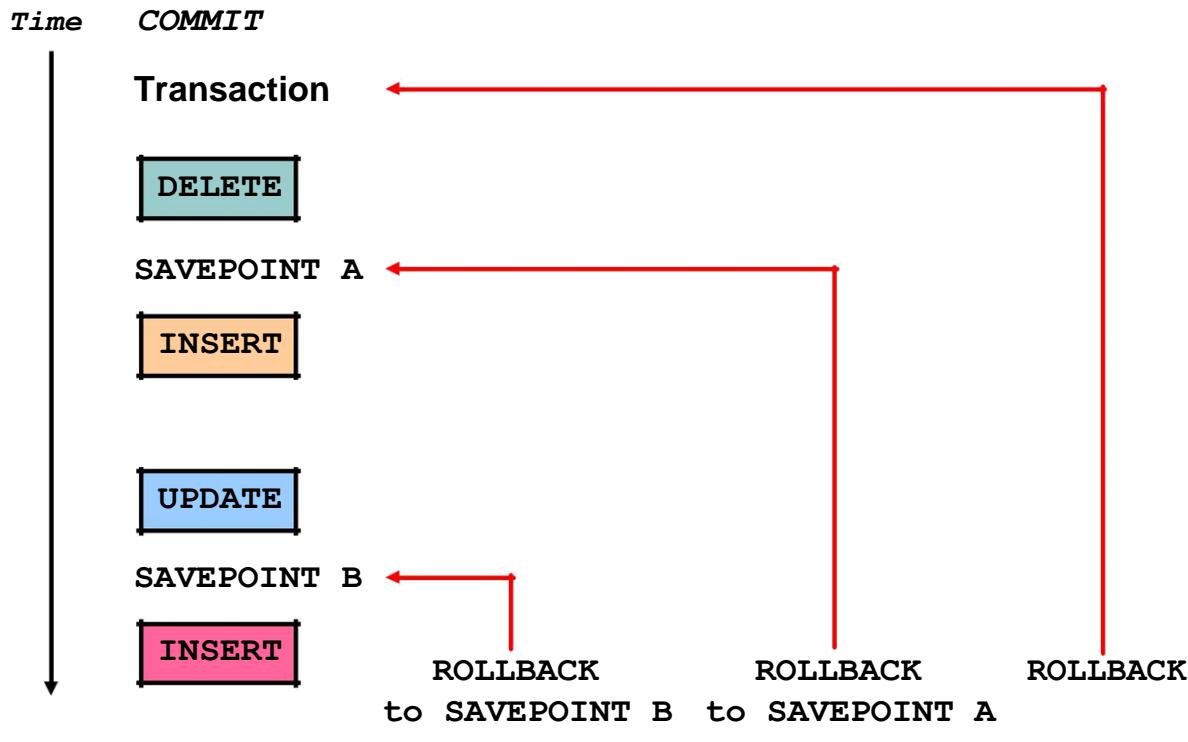


Copyright © 2009, Oracle. All rights reserved.

### **Advantages of COMMIT and ROLLBACK**

With the COMMIT and ROLLBACK statements, you have control over making changes to the data permanent.

# Controlling Transactions



**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

## Explicit Transaction Control Statements

You can control the logic of transactions by using the COMMIT, SAVEPOINT, and ROLLBACK statements.

| Statement                            | Description                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| COMMIT                               | Ends the current transaction by making all pending data changes permanent                                                                                                                                                                                                                                                                                                                               |
| SAVEPOINT <i>name</i>                | Marks a savepoint within the current transaction                                                                                                                                                                                                                                                                                                                                                        |
| ROLLBACK                             | ROLLBACK ends the current transaction by discarding all pending data changes.                                                                                                                                                                                                                                                                                                                           |
| ROLLBACK TO<br>SAVEPOINT <i>name</i> | ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding any changes and or savepoints that were created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created. |

**Note:** SAVEPOINT is not ANSI standard SQL.

## Rolling Back Changes to a Marker

- Create a marker in a current transaction by using the `SAVEPOINT` statement.
- Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```
UPDATE...
SAVEPOINT update_done;
SAVEPOINT update_done succeeded
INSERT...
ROLLBACK TO update_done;
ROLLBACK TO succeeded
```



Copyright © 2009, Oracle. All rights reserved.

### Rolling Back Changes to a Marker

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

If you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

## Implicit Transaction Processing

- An automatic commit occurs under the following circumstances:
  - DDL statement is issued
  - DCL statement is issued
  - Normal exit from SQL Developer or SQL\*Plus, without explicitly issuing COMMIT or ROLLBACK statements
- An automatic rollback occurs under an abnormal termination of SQL Developer or SQL\*Plus or a system failure.



Copyright © 2009, Oracle. All rights reserved.

## Implicit Transaction Processing

| Status             | Circumstances                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Automatic commit   | DDL statement or DCL statement is issued. SQL Developer or SQL*Plus exited normally, without explicitly issuing COMMIT or ROLLBACK commands. |
| Automatic rollback | Abnormal termination of SQL Developer or SQL*Plus or system failure                                                                          |

**Note:** In SQL\*Plus, the AUTOCOMMIT command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the COMMIT statement can still be issued explicitly. Also, the COMMIT statement is issued when a DDL statement is issued or when you exit SQL\*Plus. The SET AUTOCOMMIT ON/OFF command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled. To enable Autocommit, perform the following:

- In the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.
- On the right pane, check the Autocommit in SQL Worksheet option. Click OK.

## **Implicit Transaction Processing (continued)**

### **System Failures**

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to their state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL\*Plus, a normal exit is accomplished by entering the EXIT command at the prompt.

Closing the window is interpreted as an abnormal exit.

## State of the Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the `SELECT` statement.
- Other users *cannot* view the results of the DML statements by the current user.
- The affected rows are *locked*; other users cannot change the data in the affected rows.

The red horizontal bar spans most of the page width, centered below the main content area.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Committing Changes

Every data change made during the transaction is temporary until the transaction is committed. The state of the data before COMMIT or ROLLBACK statements are issued can be described as follows:

- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit.
- The affected rows are locked; other users cannot change the data in the affected rows.

## State of the Data After COMMIT

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.

The red horizontal bar spans most of the page width, centered below the main content area.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Committing Changes (continued)

Make all pending changes permanent by using the COMMIT statement. Here is what happens after a COMMIT statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All users can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
- All savepoints are erased.

# Committing Data

- Make the changes:

```
DELETE FROM employees
WHERE employee_id = 99999;
1 rows deleted

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL,
1700); 1 rows inserted
```

- Commit the changes:

```
COMMIT;
Commit complete
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Committing Changes (continued)

The slide example deletes a row from the EMPLOYEES table and inserts a new row into the DEPARTMENTS table. It then makes the change permanent by issuing the COMMIT statement.

### Example

Remove departments 290 and 300 in the DEPARTMENTS table, and update a row in the EMPLOYEES table. Make the data change permanent.

```
DELETE FROM departments
WHERE department_id IN (290, 300);
1 rows deleted.
```

```
UPDATE employees
SET department_id = 80
WHERE employee_id = 206;
1 rows updated.
```

```
COMMIT;
Commit Complete.
```

## State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;
20 rows deleted
ROLLBACK;
Rollback complete
```



Copyright © 2009, Oracle. All rights reserved.

### Rolling Back Changes

Discard all pending changes by using the ROLLBACK statement, which results in the following:

- Data changes are undone.
- The previous state of the data is restored.
- Locks on the affected rows are released.

## State of the Data After ROLLBACK

```
DELETE FROM test;
25,000 rows deleted

ROLLBACK;
Rollback complete

DELETE FROM test WHERE id = 100;
1 rows deleted

SELECT * FROM test WHERE id = 100;
No rows selected

COMMIT;
Commit complete
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Example

While attempting to remove a record from the TEST table, you can accidentally empty the table. You can correct the mistake, reissue the proper statement, and make the data change permanent.

## Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.

The red horizontal bar spans most of the page width, centered below the ORACLE logo.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Statement-Level Rollback

Part of a transaction can be discarded by an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the transaction are not discarded. They can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your transactions explicitly by executing a COMMIT or ROLLBACK statement.

## Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with changes made by another user.
- Read consistency ensures that on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers



Copyright © 2009, Oracle. All rights reserved.

### Read Consistency

Database users access the database in two ways:

- Read operations (SELECT statement)
- Write operations (INSERT, UPDATE, DELETE statements)

You need read consistency so that the following occur:

- The database reader and writer are ensured a consistent view of the data.
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent way.
- Changes made by one writer do not disrupt or conflict with changes that another writer is making.

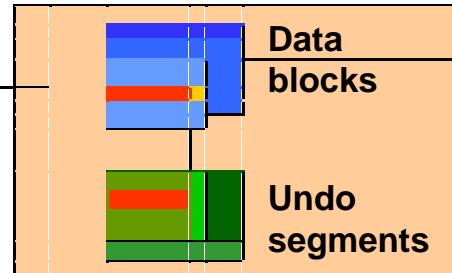
The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

# Implementation of Read Consistency

User A



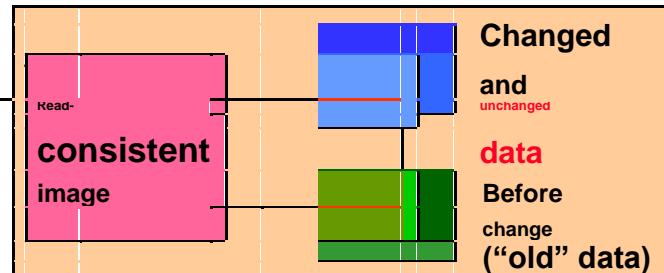
```
UPDATE employees
SET salary = 7000
WHERE last_name = 'Grant';
```



User B



```
SELECT *
FROM userA.employees;
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Implementation of Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in undo segments. The read-consistent image is constructed from committed data from the table and old data being changed and not yet committed from the undo segment.

When an insert, update, or delete operation is made to the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, still see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

Before changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a select statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:

- The original, older version of the data in the undo segment is written back to the table.
- All users see the database as it existed before the transaction began.

## Summary

In this lesson, you should have learned how to use the following statements:

| Function  | Description                                  |
|-----------|----------------------------------------------|
| INSERT    | Adds a new row to the table                  |
| UPDATE    | Modifies existing rows in the table          |
| DELETE    | Removes existing rows from the table         |
| COMMIT    | Makes all pending changes permanent          |
| SAVEPOINT | Is used to roll back to the savepoint marker |
| ROLLBACK  | Discards all pending data changes            |



Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to manipulate data in the Oracle Database by using the `INSERT`, `UPDATE`, and `DELETE` statements, as well as how to control data changes by using the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

The Oracle server guarantees a consistent view of data at all times.

## Practice 8: Overview

This practice covers the following topics:

- Inserting rows into tables
- Updating and deleting rows in a table
- Controlling transactions



Copyright © 2009, Oracle. All rights reserved.

### Practice 8: Overview

In this practice, you add rows to the MY\_EMPLOYEE table, update and delete data from the table, and control your transactions.

## Practice 8

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, you use the `MY_EMPLOYEE` table before giving the statements to the HR department.

### Insert data into the `MY_EMPLOYEE` table.

- Run the statement in the `lab_08_01.sql` script to build the `MY_EMPLOYEE` table to be used for the lab.
- Describe the structure of the `MY_EMPLOYEE` table to identify the column names.

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| ID              | NOT NULL | NUMBER(4)    |
| LAST_NAME       |          | VARCHAR2(25) |
| FIRST_NAME      |          | VARCHAR2(25) |
| USERID          |          | VARCHAR2(8)  |
| SALARY          |          | NUMBER(9,2)  |
| 5 rows selected |          |              |

- Create an `INSERT` statement to add *the first row* of data to the `MY_EMPLOYEE` table from the following sample data. Do not list the columns in the `INSERT` clause. *Do not enter all rows yet.*

| ID | LAST_NAME | FIRST_NAME | USERID   | SALARY |
|----|-----------|------------|----------|--------|
| 1  | Patel     | Ralph      | rpatel   | 895    |
| 2  | Dancs     | Betty      | bdancs   | 860    |
| 3  | Biri      | Ben        | bbiri    | 1100   |
| 4  | Newman    | Chad       | cnewman  | 750    |
| 5  | Ropeburn  | Audrey     | aropebur | 1550   |

- Populate the `MY_EMPLOYEE` table with the second row of sample data from the preceding list. This time, list the columns explicitly in the `INSERT` clause.
- Confirm your addition to the table.

| ID | LAST_NAME | FIRST_NAME | USERID | SALARY |
|----|-----------|------------|--------|--------|
| 1  | Patel     | Ralph      | rpatel | 895    |
| 2  | Dancs     | Betty      | bdancs | 860    |

- Write an `INSERT` statement in a dynamic reusable script file named `loademp.sql` to load rows into the `MY_EMPLOYEE` table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID. Save this script to a file named `lab_08_06.sql`.
- Populate the table with the next two rows of sample data listed in step 3 by running the `INSERT` statement in the script that you created.

## Practice 8 (continued)

8. Confirm your additions to the table.

| ID | LAST_NAME | FIRST_NAME | USERID  | SALARY |
|----|-----------|------------|---------|--------|
| 1  | Patel     | Ralph      | rpatel  | 895    |
| 2  | Dancs     | Betty      | bdancs  | 860    |
| 3  | Biri      | Ben        | bbiri   | 1100   |
| 4  | Newman    | Chad       | cnewman | 750    |

9. Make the data additions permanent.

### Update and delete data in the MY\_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.  
11. Change the salary to \$1,000 for all employees who have a salary less than \$900.  
12. Verify your changes to the table.

| LAST_NAME | SALARY |
|-----------|--------|
| Patel     | 1000   |
| Dancs     | 1000   |
| Drexler   | 1100   |
| Newman    | 1000   |

13. Delete Betty Dancs from the MY\_EMPLOYEE table.

14. Confirm your changes to the table.

| ID | LAST_NAME | FIRST_NAME | USERID  | SALARY |
|----|-----------|------------|---------|--------|
| 1  | Patel     | Ralph      | rpatel  | 1000   |
| 2  | Drexler   | Ben        | bbiri   | 1100   |
| 3  | Newman    | Chad       | cnewman | 1000   |

15. Commit all pending changes.

## Practice 8 (continued)

### Control data transaction to the **MY\_EMPLOYEE** table.

16. Populate the table with the last row of sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.
17. Confirm your addition to the table.

| ID | LAST_NAME  | FIRST_NAME | USERID   | SALARY |
|----|------------|------------|----------|--------|
| 1  | 1 Patel    | Ralph      | rpatel   | 1000   |
| 2  | 3 Drexler  | Ben        | bbiri    | 1100   |
| 3  | 4 Newman   | Chad       | cnewman  | 1000   |
| 4  | 5 Ropeburn | Audrey     | aropebur | 1550   |

18. Mark an intermediate point in the processing of the transaction.
19. Empty the entire table.
20. Confirm that the table is empty.
21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.
22. Confirm that the new row is still intact.

| ID | LAST_NAME  | FIRST_NAME | USERID   | SALARY |
|----|------------|------------|----------|--------|
| 1  | 1 Patel    | Ralph      | rpatel   | 1000   |
| 2  | 3 Drexler  | Ben        | bbiri    | 1100   |
| 3  | 4 Newman   | Chad       | cnewman  | 1000   |
| 4  | 5 Ropeburn | Audrey     | aropebur | 1550   |

23. Make the data addition permanent.

Oracle Internal & Oracle Academy Use Only