

Oracle Database 11g: SQL Fundamentals I

Volume I • Student Guide

D49996GC20

Edition 2.0

October 2009

D63147

ORACLE®

Authors

Salome Clement
Brian Pottle
Puja Singh

Technical Contributors and Reviewers

Anjulaponni Azhagulekshmi
Clair Bennett
Zarko Cesljas
Yanti Chang
Gerlinde Frenzen
Steve Friedberg
Joel Goodman
Nancy Greenberg
Pedro Neves
Surya Rekha
Helen Robertson
Lauran Serhal
Tulika Srivastava

Editors

Aju Kumar
Arijit Ghosh

Graphic Designer

Rajiv Chandrabhanu

Publishers

Pavithran Adka
Veena Narasimhan

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

I Introduction

Lesson Objectives	I-2
Lesson Agenda	I-3
Course Objectives	I-4
Course Agenda	I-5
Appendixes Used in the Course	I-7
Lesson Agenda	I-8
Oracle Database 11g: Focus Areas	I-9
Oracle Database 11g	I-10
Oracle Fusion Middleware	I-12
Oracle Enterprise Manager Grid Control	I-13
Oracle BI Publisher	I-14
Lesson Agenda	I-15
Relational and Object Relational Database Management Systems	I-16
Data Storage on Different Media	I-17
Relational Database Concept	I-18
Definition of a Relational Database	I-19
Data Models	I-20
Entity Relationship Model	I-21
Entity Relationship Modeling Conventions	I-23
Relating Multiple Tables	I-25
Relational Database Terminology	I-27
Lesson Agenda	I-29
Using SQL to Query Your Database	I-30
SQL Statements	I-31
Development Environments for SQL	I-32
Lesson Agenda	I-33
Human Resources (HR) Schema	I-34
Tables Used in the Course	I-35
Lesson Agenda	I-36
Oracle Database 11g Documentation	I-37
Additional Resources	I-38
Summary	I-39
Practice I: Overview	I-40

1 Retrieving Data Using the SQL `SELECT` Statement

Objectives 1-2

Lesson Agenda 1-3

Capabilities of SQL `SELECT` Statements 1-4

Basic `SELECT` Statement 1-5

Selecting All Columns 1-6

Selecting Specific Columns 1-7

Writing SQL Statements 1-8

Column Heading Defaults 1-9

Lesson Agenda 1-10

Arithmetic Expressions 1-11

Using Arithmetic Operators 1-12

Operator Precedence 1-13

Defining a Null Value 1-14

Null Values in Arithmetic Expressions 1-15

Lesson Agenda 1-16

Defining a Column Alias 1-17

Using Column Aliases 1-18

Lesson Agenda 1-19

Concatenation Operator 1-20

Literal Character Strings 1-21

Using Literal Character Strings 1-22

Alternative Quote (`q`) Operator 1-23

Duplicate Rows 1-24

Lesson Agenda 1-25

Displaying the Table Structure 1-26

Using the `DESCRIBE` Command 1-27

Quiz 1-28

Summary 1-29

Practice 1: Overview 1-30

2 Restricting and Sorting Data

Objectives 2-2

Lesson Agenda 2-3

Limiting Rows Using a Selection 2-4

Limiting the Rows That Are Selected 2-5

Using the `WHERE` Clause 2-6

Character Strings and Dates 2-7

Comparison Operators 2-8

Using Comparison Operators 2-9

- Range Conditions Using the `BETWEEN` Operator 2-10
- Membership Condition Using the `IN` Operator 2-11
- Pattern Matching Using the `LIKE` Operator 2-12
- Combining Wildcard Characters 2-13
- Using the `NULL` Conditions 2-14
- Defining Conditions Using the Logical Operators 2-15
- Using the `AND` Operator 2-16
- Using the `OR` Operator 2-17
- Using the `NOT` Operator 2-18
- Lesson Agenda 2-19
- Rules of Precedence 2-20
- Lesson Agenda 2-22
- Using the `ORDER BY` Clause 2-23
- Sorting 2-24
- Lesson Agenda 2-26
- Substitution Variables 2-27
- Using the Single-Ampersand Substitution Variable 2-29
- Character and Date Values with Substitution Variables 2-31
- Specifying Column Names, Expressions, and Text 2-32
- Using the Double-Ampersand Substitution Variable 2-33
- Lesson Agenda 2-34
- Using the `DEFINE` Command 2-35
- Using the `VERIFY` Command 2-36
- Quiz 2-37
- Summary 2-38
- Practice 2: Overview 2-39

3 Using Single-Row Functions to Customize Output

- Objectives 3-2
- Lesson Agenda 3-3
- SQL Functions 3-4
- Two Types of SQL Functions 3-5
- Single-Row Functions 3-6
- Lesson Agenda 3-8
- Character Functions 3-9
- Case-Conversion Functions 3-11
- Using Case-Conversion Functions 3-12
- Character-Manipulation Functions 3-13
- Using the Character-Manipulation Functions 3-14
- Lesson Agenda 3-15

Number Functions 3-16
Using the `ROUND` Function 3-17
Using the `TRUNC` Function 3-18
Using the `MOD` Function 3-19
Lesson Agenda 3-20
Working with Dates 3-21
RR Date Format 3-22
Using the `SYSDATE` Function 3-24
Arithmetic with Dates 3-25
Using Arithmetic Operators with Dates 3-26
Lesson Agenda 3-27
Date-Manipulation Functions 3-28
Using Date Functions 3-29
Using `ROUND` and `TRUNC` Functions with Dates 3-30
Quiz 3-31
Summary 3-32
Practice 3: Overview 3-33

4 Using Conversion Functions and Conditional Expressions

Objectives 4-2
Lesson Agenda 4-3
Conversion Functions 4-4
Implicit Data Type Conversion 4-5
Explicit Data Type Conversion 4-7
Lesson Agenda 4-10
Using the `TO_CHAR` Function with Dates 4-11
Elements of the Date Format Model 4-12
Using the `TO_CHAR` Function with Dates 4-16
Using the `TO_CHAR` Function with Numbers 4-17
Using the `TO_NUMBER` and `TO_DATE` Functions 4-20
Using the `TO_CHAR` and `TO_DATE` Function with the RR Date Format 4-22
Lesson Agenda 4-23
Nesting Functions 4-24
Nesting Functions: Example 1 4-25
Nesting Functions: Example 2 4-26
Lesson Agenda 4-27
General Functions 4-28
NVL Function 4-29
Using the `NVL` Function 4-30
Using the `NVL2` Function 4-31

Using the <code>NULLIF</code> Function	4-32
Using the <code>COALESCE</code> Function	4-33
Lesson Agenda	4-36
Conditional Expressions	4-37
<code>CASE</code> Expression	4-38
Using the <code>CASE</code> Expression	4-39
<code>DECODE</code> Function	4-40
Using the <code>DECODE</code> Function	4-41
Quiz	4-43
Summary	4-44
Practice 4: Overview	4-45
5 Reporting Aggregated Data Using the Group Functions	
Objectives	5-2
Lesson Agenda	5-3
What Are Group Functions?	5-4
Types of Group Functions	5-5
Group Functions: Syntax	5-6
Using the <code>AVG</code> and <code>SUM</code> Functions	5-7
Using the <code>MIN</code> and <code>MAX</code> Functions	5-8
Using the <code>COUNT</code> Function	5-9
Using the <code>DISTINCT</code> Keyword	5-10
Group Functions and Null Values	5-11
Lesson Agenda	5-12
Creating Groups of Data	5-13
Creating Groups of Data: <code>GROUP BY</code> Clause Syntax	5-14
Using the <code>GROUP BY</code> Clause	5-15
Grouping by More Than One Column	5-17
Using the <code>GROUP BY</code> Clause on Multiple Columns	5-18
Illegal Queries Using Group Functions	5-19
Restricting Group Results	5-21
Restricting Group Results with the <code>HAVING</code> Clause	5-22
Using the <code>HAVING</code> Clause	5-23
Lesson Agenda	5-25
Nesting Group Functions	5-26
Quiz	5-27
Summary	5-28
Practice 5: Overview	5-29

6 Displaying Data from Multiple Tables Using Joins

Objectives 6-2

Lesson Agenda 6-3

Obtaining Data from Multiple Tables 6-4

Types of Joins 6-5

Joining Tables Using SQL: 1999 Syntax 6-6

Qualifying Ambiguous Column Names 6-7

Lesson Agenda 6-8

Creating Natural Joins 6-9

Retrieving Records with Natural Joins 6-10

Creating Joins with the `USING` Clause 6-11

Joining Column Names 6-12

Retrieving Records with the `USING` Clause 6-13

Using Table Aliases with the `USING` Clause 6-14

Creating Joins with the `ON` Clause 6-15

Retrieving Records with the `ON` Clause 6-16

Creating Three-Way Joins with the `ON` Clause 6-17

Applying Additional Conditions to a Join 6-18

Lesson Agenda 6-19

Joining a Table to Itself 6-20

Self-Joins Using the `ON` Clause 6-21

Lesson Agenda 6-22

Nonequijoins 6-23

Retrieving Records with Nonequijoins 6-24

Lesson Agenda 6-25

Returning Records with No Direct Match Using `OUTER` Joins 6-26

`INNER` Versus `OUTER` Joins 6-27

`LEFT OUTER JOIN` 6-28

`RIGHT OUTER JOIN` 6-29

`FULL OUTER JOIN` 6-30

Lesson Agenda 6-31

Cartesian Products 6-32

Generating a Cartesian Product 6-33

Creating Cross Joins 6-34

Quiz 6-35

Summary 6-36

Practice 6: Overview 6-37

7 Using Subqueries to Solve Queries

Objectives 7-2

Lesson Agenda 7-3

Using a Subquery to Solve a Problem 7-4

Subquery Syntax 7-5

Using a Subquery 7-6

Guidelines for Using Subqueries 7-7

Types of Subqueries 7-8

Lesson Agenda 7-9

Single-Row Subqueries 7-10

Executing Single-Row Subqueries 7-11

Using Group Functions in a Subquery 7-12

HAVING Clause with Subqueries 7-13

What Is Wrong with This Statement? 7-14

No Rows Returned by the Inner Query 7-15

Lesson Agenda 7-16

Multiple-Row Subqueries 7-17

Using the ANY Operator in Multiple-Row Subqueries 7-18

Using the ALL Operator in Multiple-Row Subqueries 7-19

Using the EXISTS Operator 7-20

Lesson Agenda 7-21

Null Values in a Subquery 7-22

Quiz 7-24

Summary 7-25

Practice 7: Overview 7-26

8 Using the Set Operators

Objectives 8-2

Lesson Agenda 8-3

Set Operators 8-4

Set Operator Guidelines 8-5

Oracle Server and Set Operators 8-6

Lesson Agenda 8-7

Tables Used in This Lesson 8-8

Lesson Agenda 8-12

UNION Operator 8-13

Using the UNION Operator 8-14

UNION ALL Operator 8-16

Using the UNION ALL Operator 8-17

Lesson Agenda 8-18

- INTERSECT Operator 8-19
- Using the INTERSECT Operator 8-20
- Lesson Agenda 8-21
- MINUS Operator 8-22
- Using the MINUS Operator 8-23
- Lesson Agenda 8-24
- Matching the SELECT Statements 8-25
- Matching the SELECT Statement: Example 8-26
- Lesson Agenda 8-27
- Using the ORDER BY Clause in Set Operations 8-28
- Quiz 8-29
- Summary 8-30
- Practice 8: Overview 8-31

9 Manipulating Data

- Objectives 9-2
- Lesson Agenda 9-3
- Data Manipulation Language 9-4
- Adding a New Row to a Table 9-5
- INSERT Statement Syntax 9-6
- Inserting New Rows 9-7
- Inserting Rows with Null Values 9-8
- Inserting Special Values 9-9
- Inserting Specific Date and Time Values 9-10
- Creating a Script 9-11
- Copying Rows from Another Table 9-12
- Lesson Agenda 9-13
- Changing Data in a Table 9-14
- UPDATE Statement Syntax 9-15
- Updating Rows in a Table 9-16
- Updating Two Columns with a Subquery 9-17
- Updating Rows Based on Another Table 9-18
- Lesson Agenda 9-19
- Removing a Row from a Table 9-20
- DELETE Statement 9-21
- Deleting Rows from a Table 9-22
- Deleting Rows Based on Another Table 9-23
- TRUNCATE Statement 9-24
- Lesson Agenda 9-25
- Database Transactions 9-26

Database Transactions: Start and End	9-27
Advantages of <code>COMMIT</code> and <code>ROLLBACK</code> Statements	9-28
Explicit Transaction Control Statements	9-29
Rolling Back Changes to a Marker	9-30
Implicit Transaction Processing	9-31
State of the Data Before <code>COMMIT</code> or <code>ROLLBACK</code>	9-33
State of the Data After <code>COMMIT</code>	9-34
Committing Data	9-35
State of the Data After <code>ROLLBACK</code>	9-36
State of the Data After <code>ROLLBACK</code> : Example	9-37
Statement-Level Rollback	9-38
Lesson Agenda	9-39
Read Consistency	9-40
Implementing Read Consistency	9-41
Lesson Agenda	9-42
<code>FOR UPDATE</code> Clause in a <code>SELECT</code> Statement	9-43
<code>FOR UPDATE</code> Clause: Examples	9-44
Quiz	9-46
Summary	9-47
Practice 9: Overview	9-48

10 Using DDL Statements to Create and Manage Tables

Objectives	10-2
Lesson Agenda	10-3
Database Objects	10-4
Naming Rules	10-5
Lesson Agenda	10-6
<code>CREATE TABLE</code> Statement	10-7
Referencing Another User's Tables	10-8
<code>DEFAULT</code> Option	10-9
Creating Tables	10-10
Lesson Agenda	10-11
Data Types	10-12
Datetime Data Types	10-14
Lesson Agenda	10-15
Including Constraints	10-16
Constraint Guidelines	10-17
Defining Constraints	10-18
<code>NOT NULL</code> Constraint	10-20
<code>UNIQUE</code> Constraint	10-21

PRIMARY KEY Constraint 10-23
FOREIGN KEY Constraint 10-24
FOREIGN KEY Constraint: Keywords 10-26
CHECK Constraint 10-27
CREATE TABLE: Example 10-28
Violating Constraints 10-29
Lesson Agenda 10-31
Creating a Table Using a Subquery 10-32
Lesson Agenda 10-34
ALTER TABLE Statement 10-35
Read-Only Tables 10-36
Lesson Agenda 10-37
Dropping a Table 10-38
Quiz 10-39
Summary 10-40
Practice 10: Overview 10-41

11 Creating Other Schema Objects

Objectives 11-2
Lesson Agenda 11-3
Database Objects 11-4
What Is a View? 11-5
Advantages of Views 11-6
Simple Views and Complex Views 11-7
Creating a View 11-8
Retrieving Data from a View 11-11
Modifying a View 11-12
Creating a Complex View 11-13
Rules for Performing DML Operations on a View 11-14
Using the WITH CHECK OPTION Clause 11-17
Denying DML Operations 11-18
Removing a View 11-20
Practice 11: Overview of Part 1 11-21
Lesson Agenda 11-22
Sequences 11-23
CREATE SEQUENCE Statement: Syntax 11-25
Creating a Sequence 11-26
NEXTVAL and CURRVAL Pseudocolumns 11-27
Using a Sequence 11-29
Caching Sequence Values 11-30

Modifying a Sequence	11-31
Guidelines for Modifying a Sequence	11-32
Lesson Agenda	11-33
Indexes	11-34
How Are Indexes Created?	11-36
Creating an Index	11-37
Index Creation Guidelines	11-38
Removing an Index	11-39
Lesson Agenda	11-40
Synonyms	11-41
Creating a Synonym for an Object	11-42
Creating and Removing Synonyms	11-43
Quiz	11-44
Summary	11-45
Practice 11: Overview of Part 2	11-46

Appendix A: Practice Solutions

Appendix B: Table Descriptions

Appendix C: Using SQL Developer

Objectives	C-2
What Is Oracle SQL Developer?	C-3
Specifications of SQL Developer	C-4
SQL Developer 1.5 Interface	C-5
Creating a Database Connection	C-7
Browsing Database Objects	C-10
Displaying the Table Structure	C-11
Browsing Files	C-12
Creating a Schema Object	C-13
Creating a New Table: Example	C-14
Using the SQL Worksheet	C-15
Executing SQL Statements	C-18
Saving SQL Scripts	C-19
Executing Saved Script Files: Method 1	C-20
Executing Saved Script Files: Method 2	C-21
Formatting the SQL Code	C-22
Using Snippets	C-23
Using Snippets: Example	C-24
Debugging Procedures and Functions	C-25
Database Reporting	C-26

Creating a User-Defined Report C-27
Search Engines and External Tools C-28
Setting Preferences C-29
Resetting the SQL Developer Layout C-30
Summary C-31

Appendix D: Using SQL*Plus

Objectives D-2
SQL and SQL*Plus Interaction D-3
SQL Statements Versus SQL*Plus Commands D-4
Overview of SQL*Plus D-5
Logging In to SQL*Plus D-6
Displaying the Table Structure D-7
SQL*Plus Editing Commands D-9
Using LIST, n, and APPEND D-11
Using the CHANGE Command D-12
SQL*Plus File Commands D-13
Using the SAVE, START Commands D-14
SERVEROUTPUT Command D-15
Using the SQL*Plus SPOOL Command D-16
Using the AUTOTRACE Command D-17
Summary D-18

Appendix E: Using JDeveloper

Objectives E-2
Oracle JDeveloper E-3
Database Navigator E-4
Creating Connection E-5
Browsing Database Objects E-6
Executing SQL Statements E-7
Creating Program Units E-8
Compiling E-9
Running a Program Unit E-10
Dropping a Program Unit E-11
Structure Window E-12
Editor Window E-13
Application Navigator E-14
Deploying Java Stored Procedures E-15

Publishing Java to PL/SQL	E-16
How Can I Learn More About JDeveloper 11g?	E-17
Summary	E-18

Appendix F: Oracle Join Syntax

Objectives	F-2
Obtaining Data from Multiple Tables	F-3
Cartesian Products	F-4
Generating a Cartesian Product	F-5
Types of Oracle-Proprietary Joins	F-6
Joining Tables Using Oracle Syntax	F-7
Qualifying Ambiguous Column Names	F-8
Equijoins	F-9
Retrieving Records with Equijoins	F-10
Retrieving Records with Equijoins: Example	F-11
Additional Search Conditions Using the <code>AND</code> Operator	F-12
Joining More than Two Tables	F-13
Nonequijoins	F-14
Retrieving Records with Nonequijoins	F-15
Returning Records with No Direct Match with Outer Joins	F-16
Outer Joins: Syntax	F-17
Using Outer Joins	F-18
Outer Join: Another Example	F-19
Joining a Table to Itself	F-20
Self-Join: Example	F-21
Summary	F-22
Practice F: Overview	F-23

Additional Practices and Solutions

Index

I Introduction

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Lesson Objectives

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

- Define the goals of the course
- List the features of Oracle Database 11g
- Discuss the theoretical and physical aspects of a relational database
- Describe Oracle server's implementation of RDBMS and object relational database management system (ORDBMS)
- Identify the development environments that can be used for this course
- Describe the database and schema used in this course

ORACLE

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 Oracle SQL Developer and SQL*Plus as development environments used for executing SQL statements, and for formatting and reporting purposes.

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 11g and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- The HR schema and the tables used in this course
- Oracle Database 11g documentation and additional resources

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Course Objectives

After completing this course, you should be able to:

- Identify the major components of Oracle Database 11g
- 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 complex queries to retrieve data from multiple tables
- Run data manipulation language (DML) statements to update data in Oracle Database 11g
- Run data definition language (DDL) statements to create and manage schema objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Course Objectives

This course offers you an introduction to the Oracle Database 11g 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.

Course Agenda

- Day 1:
 - Introduction
 - Retrieving Data Using the SQL `SELECT` Statement
 - Restricting and Sorting Data
 - Using Single-Row Functions to Customize Output
 - Using Conversion Functions and Conditional Expressions
- Day 2:
 - Reporting Aggregated Data Using the Group Functions
 - Displaying Data from Multiple Tables Using Joins
 - Using Subqueries to Solve Queries
 - Using the Set Operators

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Course Agenda

- Day 3:
 - Manipulating Data
 - Using DDL Statements to Create and Manage Tables
 - Creating Other Schema Objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Appendixes Used in the Course

- Appendix A: Practices and Solutions
- Appendix B: Table Descriptions
- Appendix C: Using SQL Developer
- Appendix D: Using SQL*Plus
- Appendix E: Using JDeveloper
- Appendix F: Oracle Join Syntax
- Appendix AP: Additional Practices and Solutions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11g and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- The HR schema and the tables used in this course
- Oracle Database 11g documentation and additional resources

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g: Focus Areas



Infrastructure
Grids

Information
Management

Application
Development

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g: Focus Areas

Oracle Database 11g offers extensive features across the following focus areas:

- **Infrastructure Grids:** The Infrastructure Grid technology of Oracle enables pooling of low-cost servers and storage to form systems that deliver the highest quality of service in terms of manageability, high availability, and performance. Oracle Database 11g consolidates and extends the benefits of grid computing. Apart from taking full advantage of grid computing, Oracle Database 11g has unique change assurance features to manage changes in a controlled and cost effective manner.
- **Information Management:** Oracle Database 11g extends the existing information management capabilities in content management, information integration, and information life-cycle management areas. Oracle provides content management of advanced data types such as Extensible Markup Language (XML), text, spatial, multimedia, medical imaging, and semantic technologies.
- **Application Development:** Oracle Database 11g has capabilities to use and manage all the major application development environments such as PL/SQL, Java/JDBC, .NET and Windows, PHP, SQL Developer, and Application Express.

Oracle Database 11g



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g

Organizations need to support multiple terabytes of information for users who demand fast and secure access to business applications round the clock. The database systems must be reliable and must be able to recover quickly in the event of any kind of failure. Oracle Database 11g is designed along the following feature areas to help organizations manage infrastructure grids easily and deliver high-quality service:

- **Manageability:** By using some of the change assurance, management automation, and fault diagnostics features, the database administrators (DBAs) can increase their productivity, reduce costs, minimize errors, and maximize quality of service. Some of the useful features that promote better management are Database Replay facility, the SQL Performance Analyzer, and the Automatic SQL Tuning facility.
- **High availability:** By using the high availability features, you can reduce the risk of down time and data loss. These features improve online operations and enable faster database upgrades.

Oracle Database 11g (continued)

- **Performance:** By using capabilities such as SecureFiles, compression for online transaction processing (OLTP), Real Application Clusters (RAC) optimizations, Result Caches, and so on, you can greatly improve the performance of your database. Oracle Database 11g enables organizations to manage large, scalable, transactional, and data warehousing systems that deliver fast data access using low-cost modular storage.
- **Security:** Oracle Database 11g helps organizations protect their information with unique secure configurations, data encryption and masking, and sophisticated auditing capabilities. It delivers a secure and scalable platform for reliable and fast access to all types of information by using the industry-standard interfaces.
- **Information integration:** Oracle Database 11g has many features to better integrate data throughout the enterprise. It also supports advanced information life-cycle management capabilities. This helps you manage the changing data in your database.

Oracle Fusion Middleware

Portfolio of leading, standards-based, and customer-proven software products that spans a range of tools and services from Java EE and developer tools, through integration services, business intelligence, collaboration, and content management



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Fusion Middleware

Oracle Fusion Middleware is a comprehensive and well-integrated family of products that offers complete support for development, deployment, and management of Service-Oriented Architecture (SOA). SOA facilitates the development of modular business services that can be easily integrated and reused, thereby reducing development and maintenance costs, and providing higher quality of services. Oracle Fusion Middleware's pluggable architecture enables you to leverage your investments in any existing application, system, or technology. Its unbreakable core technology minimizes the disruption caused by planned or unplanned outages.

Some of the products from the Oracle Fusion Middleware family include:

- **Enterprise Application Server:** Application Server
- **Integration and Process Management:** BPEL Process Manager, Oracle Business Process Analysis Suite
- **Development Tools:** Oracle Application Development Framework, JDeveloper, SOA Suite
- **Business Intelligence:** Oracle Business Activity Monitoring, Oracle Data Integrator
- **Systems Management:** Enterprise Manager
- **Identity Management:** Oracle Identity Management
- **Content Management:** Oracle Content Database Suite
- **User Interaction:** Portal, WebCenter

Oracle Enterprise Manager Grid Control

- Efficient Oracle Fusion Middleware management
- Simplifying application and infrastructure life-cycle management
- Improved database administration and application management capabilities



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Enterprise Manager Grid Control

Spanning applications, middleware, and database management, Oracle Enterprise Manager Grid Control delivers integrated enterprise management for Oracle and non-Oracle systems.

Oracle Enterprise Manager Grid Control features advanced Oracle Fusion Middleware management capabilities for the services that business applications rely upon, including SOA, Business Activity Monitoring, and Identity Management.

- **Wide-ranging management functionality** is available for your applications including service-level management, application performance management, configuration management, and change automation
- **Built-in grid automation capabilities** means that information technology responds proactively to fluctuating demand and implements new services more quickly so that businesses can thrive.
- **In-depth diagnostics and readily available remediation** can be applied across a range of applications including custom-built applications, Oracle E-Business Suite, PeopleSoft, Siebel, Oracle Fusion Middleware, Oracle Database, and underlying infrastructure
- **Extensive life cycle management capabilities** extend grid computing by providing solutions for the entire application and infrastructure life cycle, including test, stage, and production through operations. It has simplified patch management with synchronized patching, additional operating system support, and conflict detection features.

Oracle BI Publisher

- Provides a central architecture for authoring, managing, and delivering information in secure and multiple formats
- Reduces complexity and time to develop, test, and deploy all kinds of reports
 - Financial Reports, Invoices, Sales or Purchase orders, XML, and EDI/EFT(eText documents)
- Enables flexible customizations
 - For example, a Microsoft Word document report can be generated in multiple formats, such as PDF, HTML, Excel, RTF, and so on.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle BI Publisher

Oracle Database 11g also includes Oracle BI Publisher—the enterprise reporting solution from Oracle. Oracle BI Publisher (formerly known as XML Publisher) offers the most efficient and scalable reporting solution available for complex, distributed environments.

Oracle BI Publisher reduces the high costs associated with the development, customization, and maintenance of business documents, while increasing the efficiency of reports management. By using a set of familiar desktop tools, users can create and maintain their own report formats based on data queries created by the IT staff or developers.

Oracle BI Publisher report formats can be designed using Microsoft Word or Adobe Acrobat—tools that most users are already familiar with. Oracle BI Publisher also enables you to bring in data from multiple data sources into a single output document. You can deliver reports via printer, email, or fax. You can publish your report to a portal. You can even allow users to collaboratively edit and manage reports on the Web-based Distributed Authoring and Versioning (WebDav) Web servers.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11g and related products
- **Overview of relational database management concepts and terminologies**
- Introduction to SQL and its development environments
- The HR schema and the tables used in this course
- Oracle Database 11g documentation and additional resources

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Relational and Object Relational Database Management Systems

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Relational and Object Relational Database Management Systems

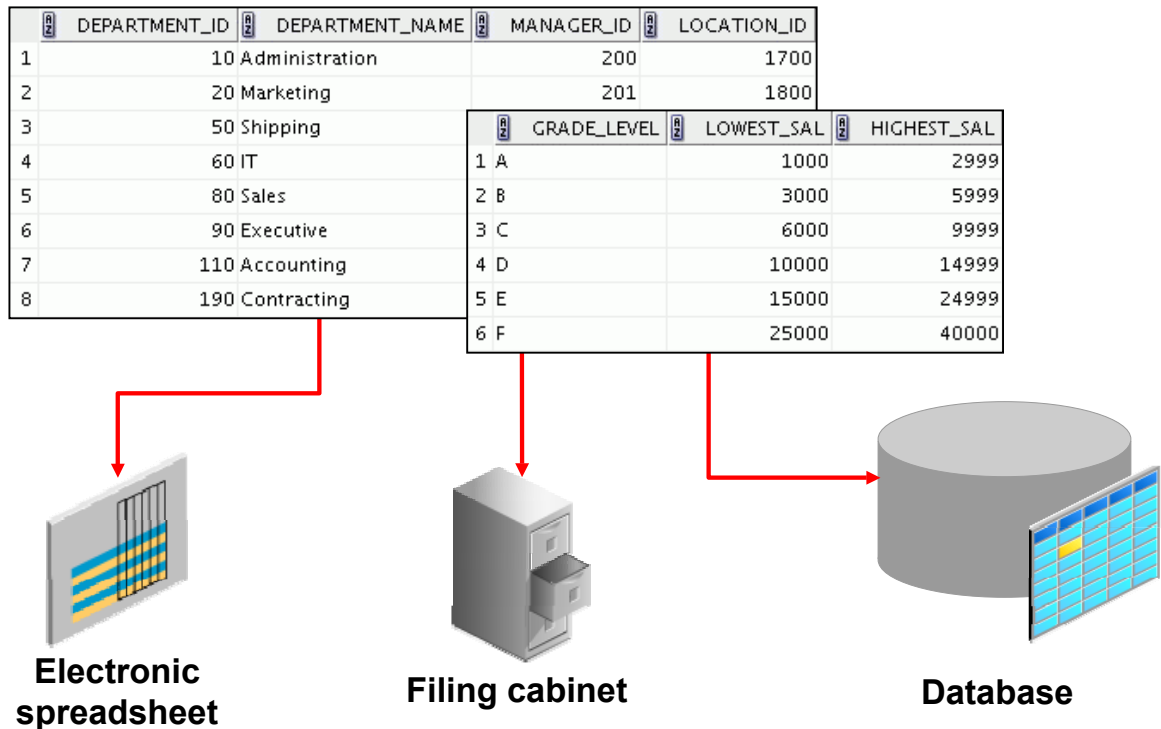
The Oracle server supports both the relational and the object relational database models.

The Oracle server extends the data-modeling capabilities to support an object relational database model that provides 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 the 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. 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, refer to *Oracle Database Concepts 11g Release 1 (11.1)*.

Data Storage on Different Media



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Data Storage on Different Media

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 its employees, departments, and salaries. These pieces of information are called *data*.

Organizations can store data in 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Relational Database Concept

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-interface products, thereby providing a total solution.

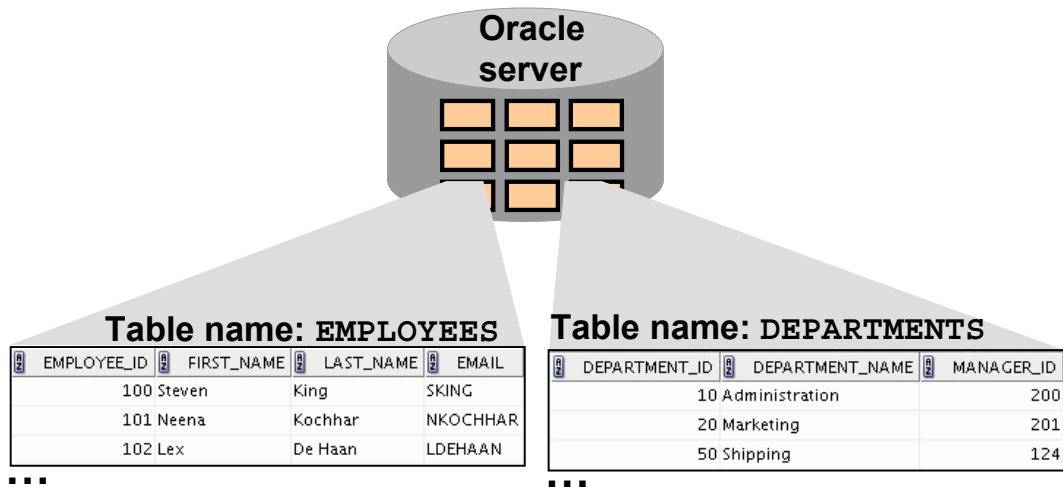
Components of the Relational Model

- 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, refer to *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

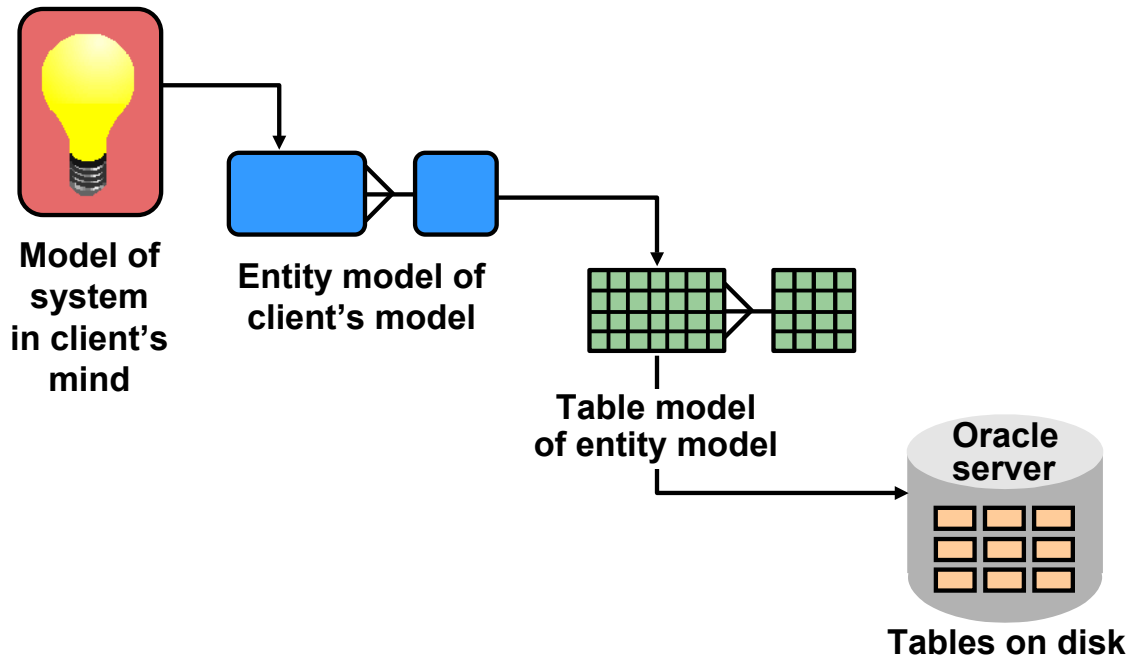
Copyright © 2009, Oracle. All rights reserved.

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Data Models

Models are the 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

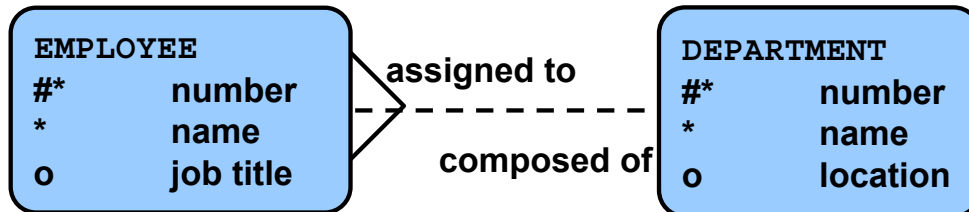
Models help to communicate the concepts that are in people's minds. They can be used to do the following:

- 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.

Entity Relationship Model

In an effective system, data is divided into discrete categories or entities. An entity relationship (ER) model is an illustration of the 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 the business. Although businesses can change their activities, the type of information tends to remain constant. Therefore, the data structures also tend to be constant.

Entity Relationship Model (continued)

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

Key Components

- **Entity:** An aspect of significance about which information must 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

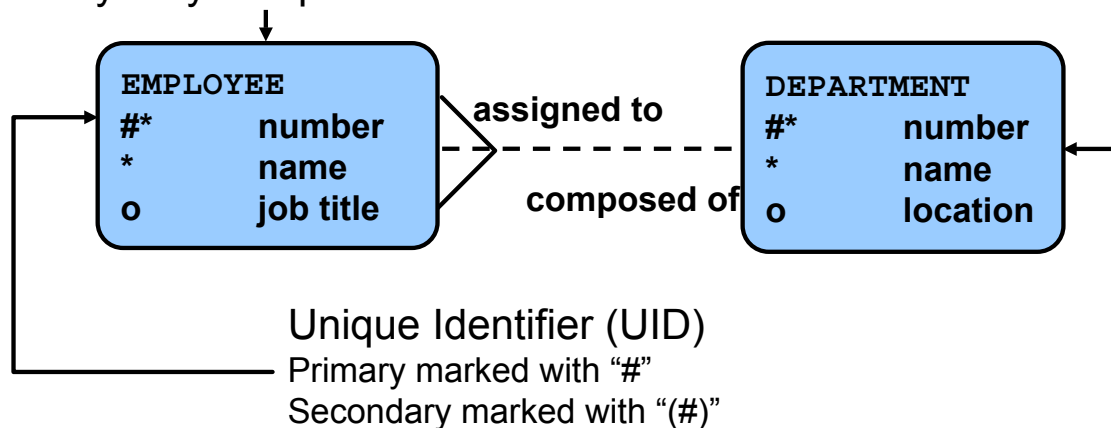
Entity Relationship Modeling Conventions

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”

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} in relation {one and only one | one or more} with the 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 hash sign “#”.
- Tag secondary UIDs with a hash sign in parentheses (#).

Relating Multiple Tables

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

Table name: EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
103	Alexander	Hunold	60
104	Bruce	Ernst	60
107	Diana	Lorentz	60
124	Kevin	Mourgos	50
141	Trenna	Rajs	50
142	Curtis	Davies	50

Primary key

Foreign key

Table name: DEPARTMENTS

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

Primary key

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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. By 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 the 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; otherwise, 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 EMPLOYEES table with the following data:

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

Annotations:

- 1: Points to a row (tuple).
- 2: Points to the EMPLOYEE_ID column header.
- 3: Points to the SALARY column header.
- 4: Points to the DEPARTMENT_ID column header.
- 5: Points to a row.
- 6: Points to the COMMISSION_PCT column header.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Relational Database Terminology

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 the 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.

Relational Database Terminology (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.

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11g and related products
- Overview of relational database management concepts and terminologies
- **Introduction to SQL and its development environments**
- The HR schema and the tables used in this course
- Oracle Database 11g documentation and additional resources

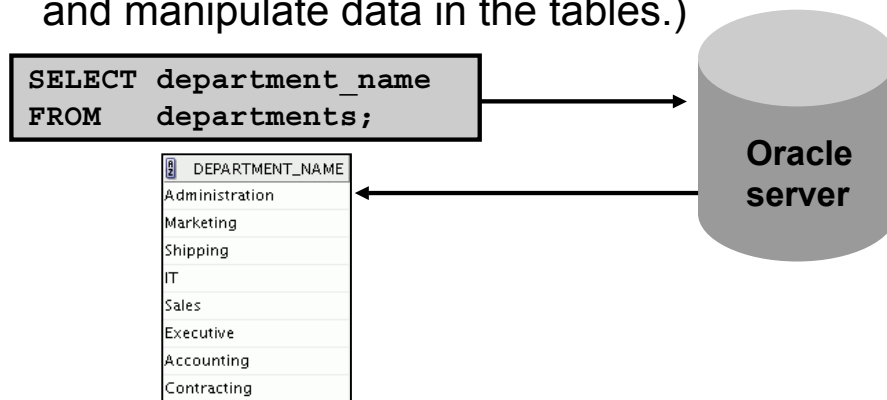
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using SQL to Query Your Database

Structured query language (SQL) is:

- The ANSI standard language for operating relational databases
- Efficient, easy to learn, and use
- Functionally complete (With SQL, you can define, retrieve, and manipulate data in the tables.)



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using SQL to Query Your 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. SQL is a set of statements with which all programs and users access data in an Oracle Database. Application programs and Oracle tools often allow users access to the database without using SQL directly, but these applications, in turn, must use SQL when executing the user's request.

SQL provides statements for a variety of tasks, including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

SQL unifies all of the preceding tasks in one consistent language and enables you to work with data at a logical level.

SQL Statements

SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Statements

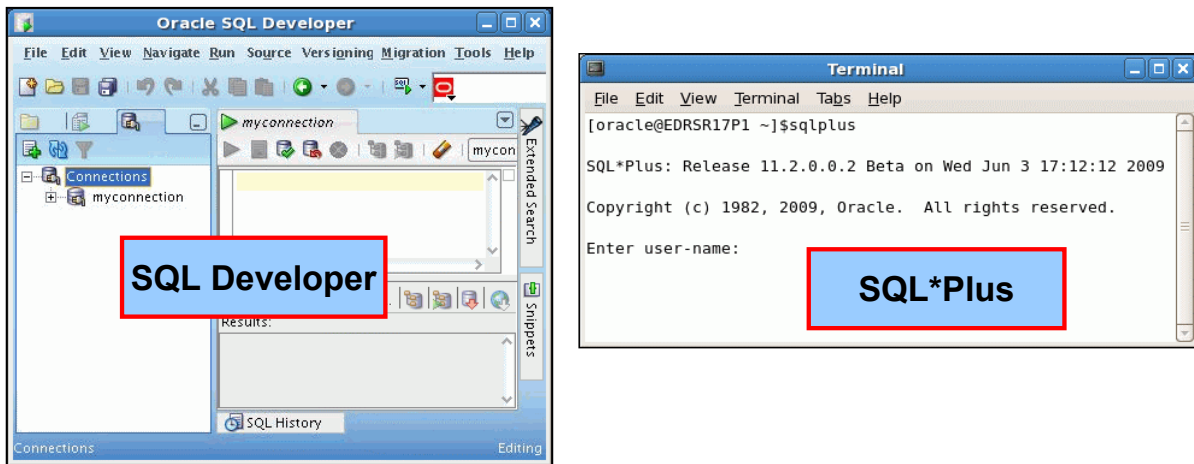
SQL statements supported by Oracle comply with industry standards. Oracle Corporation ensures future compliance with evolving standards by actively involving key personnel in SQL standards committees. The industry-accepted committees are ANSI and International Standards Organization (ISO). Both ANSI 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</i> (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Sets up, changes, and removes data structures from tables. Collectively known as <i>data definition language</i> (DDL)
GRANT REVOKE	Provides 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

Development Environments for SQL

There are two development environments for this course:

- The primary tool is Oracle SQL Developer.
- SQL*Plus command-line interface can also be used.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Development Environments for SQL

SQL Developer

This course is developed using Oracle SQL Developer as the tool for running the SQL statements discussed in the examples in the lessons and the practices. SQL Developer version 1.5.4 is shipped with Oracle Database 11g, and is the default tool for this class.

SQL*Plus

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

Note

- See Appendix C for information about using SQL Developer, including simple instructions on installing version 1.5.4.
- See Appendix D for information about using SQL*Plus.

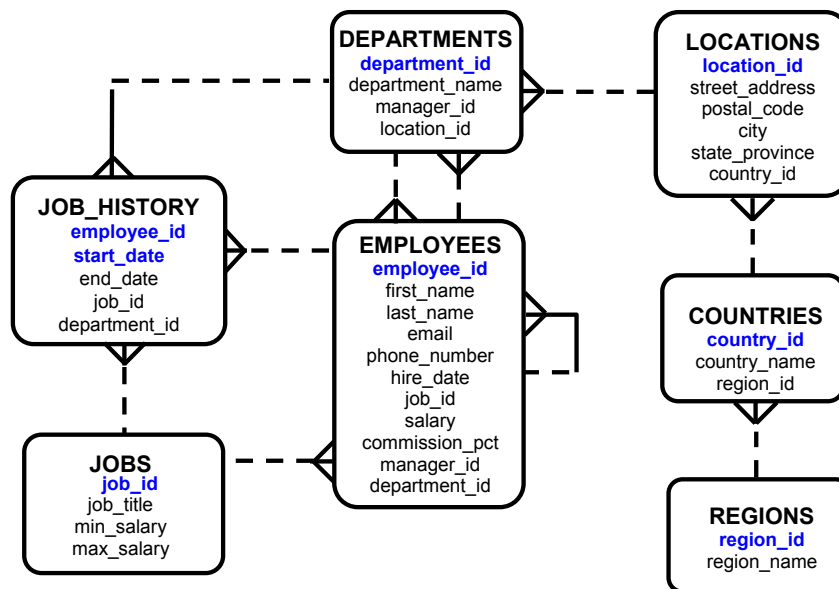
Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11g and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- **The HR schema and the tables used in this course**
- Oracle Database 11g documentation and additional resources

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Human Resources (HR) Schema



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Human Resources (HR) Schema Description

The Human Resources (HR) schema is a part of the Oracle Sample Schemas that can be installed in an Oracle Database. The practice sessions in this course use data from the HR schema.

Table Descriptions

- **REGIONS** contains rows that represent a region such as America, Asia, and so on.
- **COUNTRIES** contains rows for countries, each of which is associated with a region.
- **LOCATIONS** contains the specific address of a specific office, warehouse, or production site of a company in a particular country.
- **DEPARTMENTS** shows details about the departments in which the employees work. Each department may have a relationship representing the department manager in the **EMPLOYEES** table.
- **EMPLOYEES** contains details about each employee working for a department. Some employees may not be assigned to any department.
- **JOBS** contains the job types that can be held by each employee.
- **JOB_HISTORY** contains the job history of the employees. If an employee changes departments within a job or changes jobs within a department, a new row is inserted into this table with the earlier job information of the employee.

Tables Used in the Course

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	2600	(null)	50	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	2500	(null)	50	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	10500	0.2	80	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	11000	0.3	80	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	8600	0.2	80	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	7000	0.15	(null)	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	4400	(null)	10	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	13000	(null)	20	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	6000	(null)	20	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	12000	(null)	110	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	8300	(null)	110	WGIEZT	515.123.8181	07-JUN-94

GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

JOB_GRADES

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

DEPARTMENTS

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

Apart from these tables, you will also use the other tables listed in the previous slide such as the LOCATIONS and the JOB_HISTORY table.

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

Lesson Agenda

- Course objectives, course agenda, and appendixes used in this course
- Overview of Oracle Database 11g and related products
- Overview of relational database management concepts and terminologies
- Introduction to SQL and its development environments
- The HR schema and the tables used in this course
- Oracle Database 11g documentation and additional resources

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g Documentation

- *Oracle Database New Features Guide 11g, Release 1 (11.2)*
- *Oracle Database Reference 11g, Release 1 (11.2)*
- *Oracle Database SQL Language Reference 11g, Release 1 (11.2)*
- *Oracle Database Concepts 11g, Release 1 (11.2)*
- *Oracle Database SQL Developer User's Guide, Release 1.5*

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g Documentation

Navigate to <http://www.oracle.com/pls/db112/homepage> to access the Oracle Database 11g documentation library.

Additional Resources

For additional information about the Oracle Database 11g, refer to the following:

- *Oracle Database 11g: New Features eStudies*
- *Oracle by Example series (OBE): Oracle Database 11g*
 - http://www.oracle.com/technology/obe/11gr1_db/index.htm

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned that:

- Oracle Database 11g extends:
 - The benefits of infrastructure grids
 - The existing information management capabilities
 - The capabilities to use the major application development environments such as PL/SQL, Java/JDBC, .NET, XML, and so on
- The database is based on ORDBMS
- 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 SQL

ORACLE

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 11g with which you store and manage information by using SQL
- Oracle Fusion Middleware with which you develop, deploy, and manage modular business services that can be integrated and reused
- Oracle Enterprise Manager 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.

Practice I: Overview

This practice covers the following topics:

- Starting Oracle SQL Developer
- Creating a new database connection
- Browsing the HR tables

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice I: Overview

In this practice, you perform the following:

- Start Oracle SQL Developer and create a new connection to the `ora1` account.
- Use Oracle SQL Developer to examine data objects in the `ora1` account. The `ora1` account contains the HR schema tables.

Note the following location for the lab files:

`\home\oracle\labs\sql1\labs`

If you are asked to save any lab files, save them in 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 within 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.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

To extract data from the database, you need to use the SQL `SELECT` statement. However, you may need to restrict the columns that are displayed. This lesson describes the `SELECT` statement that is needed to perform these actions. Further, you may want to create `SELECT` statements that can be used more than once.

Lesson Agenda

- **Basic SELECT statement**
- Arithmetic expressions and NULL values in the SELECT statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the DISTINCT keyword
- DESCRIBE command

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Capabilities of SQL `SELECT` Statements

Projection

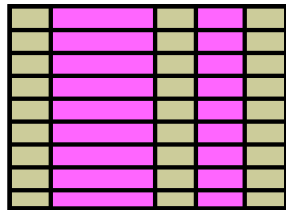


Table 1

Selection

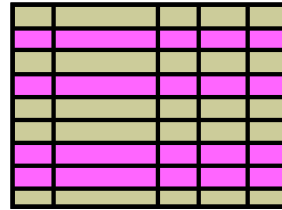


Table 1

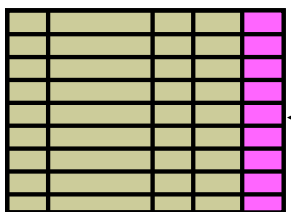


Table 1

Join

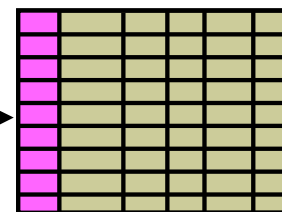


Table 2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Capabilities of SQL `SELECT` Statements

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

- **Projection:** Select the columns in a table that are returned by a query. Select as few or as many of the columns as required.
- **Selection:** Select the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.
- **Joins:** 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 Using Joins.”

Basic SELECT Statement

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

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

ORACLE

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:

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 the selected columns different headings
FROM table	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, and so on.
- A *statement* is a combination of two or more clauses—for example, SELECT * FROM employees.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Selecting All Columns

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 `DEPARTMENTS` 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;
```

Note: In SQL Developer, you can enter your SQL statement in a SQL Worksheet and click the “Execute Statement” icon or press [F9] to execute the statement. The output displayed on the Results tabbed page appears as shown in the slide.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Selecting Specific Columns

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 (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 entered 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 be optionally terminated by a semicolon (;). Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Writing SQL Statements

By using the following simple rules and guidelines, you can construct valid statements that are both easy to read and 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 names are entered in lowercase.

Executing SQL Statements

In SQL Developer, click the Run Script icon or press [F5] to run the command or commands in the SQL Worksheet. You can also click the Execute Statement icon or press [F9] to run a SQL statement in the SQL Worksheet. The Execute Statement icon executes the statement at the mouse pointer in the Enter SQL Statement box while the Run Script icon executes all the statements in the Enter SQL Statement box. The Execute Statement icon displays the output of the query on the Results tabbed page, whereas the Run Script icon emulates the SQL*Plus display and shows the output on the Script Output tabbed page.

In SQL*Plus, terminate the SQL statement with a semicolon, and then press [Enter] to run the command.

Column Heading Defaults

- SQL Developer:
 - Default heading alignment: Left-aligned
 - 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 are left-aligned.

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

	 LAST_NAME	 HIRE_DATE	 SALARY
1	Whalen	17-SEP-87	4400
2	Hartstein	17-FEB-96	13000
3	Fay	17-AUG-97	6000
4	Higgins	07-JUN-94	12000
5	Gietz	07-JUN-94	8300
6	King	17-JUN-87	24000
7	Kochhar	21-SEP-89	17000
8	De Haan	13-JAN-93	17000
9	Hunold	03-JAN-90	9000

■ ■ ■

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

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Arithmetic Expressions

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

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

ORACLE

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. All these are 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;
```

R	LAST_NAME	R	SALARY	R	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

...

ORACLE

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, evaluation is done from left to right.

You can use parentheses to force the expression that is enclosed by the 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

R	LAST_NAME	R	SALARY	R	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

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

...

ORACLE

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 with 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 $(12 * salary) + 100$ 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 the 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 with 12. Because of the parentheses, addition takes priority over multiplication.

Defining a Null Value

- Null is a value that is unavailable, unassigned, unknown, or inapplicable.
- Null is not the same as 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)
...				
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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Defining a Null Value

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

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

Note: By default, SQL Developer uses the literal, (null), to identify null values. However, you can set it to something more relevant to you. To do so, select Preferences from the Tools menu. In the Preferences dialog box, expand the Database node. Click Advanced Parameters and on the right pane, for the “Display Null value As,” enter the appropriate value.

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

...

17	Zlotkey	25200
18	Abel	39600
19	Taylor	20640
20	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 Whalen does not get any commission. Because the `COMMISSION_PCT` column in the arithmetic expression is null, the result is null.

For more information, see the section on “Basic Elements of Oracle SQL” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- **Column aliases**
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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 the alias.)
- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Defining a Column Alias

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, may be 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 blank 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

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Column Aliases

The first example displays the names and the commission percentages of all the employees. Note 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, note 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 the preceding 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. Note that the column heading in the output is exactly the same as the column alias.

Lesson Agenda

- Basic `SELECT` Statement
- Arithmetic Expressions and `NULL` values in `SELECT` statement
- Column Aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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
5	FayMK_REP
6	GietzAC_ACCOUNT

...

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 given the alias Employees. Note that the last name of the employee and the 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.

Note: You can also concatenate date expressions with other expressions or columns.

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 within 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. It 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.

The date and character literals *must* be enclosed within single quotation marks (' '); number literals need not be enclosed in a similar manner.

Using Literal Character Strings

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

	Employee 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
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Literal Character Strings

The example in the slide displays the last names and job codes of all employees. The column has the heading Employee Details. Note 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
8	De Haan: 1 Month salary = 17000

...

Alternative Quote (q) Operator

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

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

	Department and Manager
1	Administration Department's Manager Id: 200
2	Marketing Department's Manager Id: 201
3	Shipping Department's Manager Id: 124
4	IT Department's Manager Id: 103
5	Sales Department's Manager Id: 149
6	Executive Department's Manager Id: 100
7	Accounting Department's Manager Id: 205
8	Contracting Department's Manager Id:

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 select 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, brackets [] are used as the quotation mark delimiters. 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
6	80
7	10
8	60

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Duplicate Rows

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

...

Note: You may also specify the keyword `UNIQUE`, which is a synonym for the keyword `DISTINCT`.

Lesson Agenda

- Basic `SELECT` statement
- Arithmetic expressions and `NULL` values in the `SELECT` statement
- Column aliases
- Use of concatenation operator, literal character strings, alternative quote operator, and the `DISTINCT` keyword
- `DESCRIBE` command

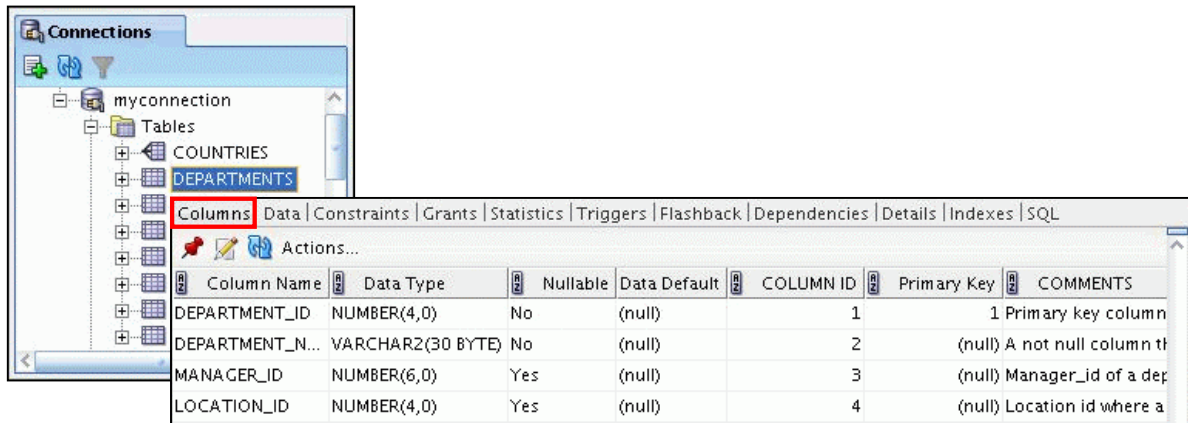
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Displaying the Table Structure

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

```
DESC[RIBE] tablename
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Displaying the Table Structure

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</i> , <i>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

Quiz

Identify the SELECT statements that execute successfully.

1. `SELECT first_name, last_name, job_id, salary*12
AS Yearly Sal
FROM employees;`

2. `SELECT first_name, last_name, job_id, salary*12
"yearly sal"
FROM employees;`

3. `SELECT first_name, last_name, job_id, salary AS
"yearly sal"
FROM employees;`

4. `SELECT first_name+last_name AS name, job_Id,
salary*12 yearly sal
FROM employees;`

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answers: 2, 3

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

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

ORACLE

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:

<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 the selected columns different headings
<code>FROM table</code>	Specifies the table containing the columns

Practice 1: Overview

This practice covers the following topics:

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 1: Overview

In this practice, you write simple `SELECT` queries. The queries cover most of the `SELECT` clauses and operations that you learned in this lesson.

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

ORACLE

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 the actions listed above.

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` conditions
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Copyright © 2009, Oracle. All rights reserved.

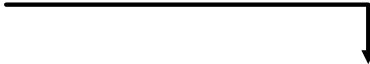
Limiting Rows Using a Selection

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

...

**“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.

ORACLE

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:

<code>WHERE</code>	Restricts the query to rows that meet a condition
--------------------	---

<i>condition</i>	Is composed of column names, expressions, constants, and a comparison operator. A condition specifies a combination of one or more expressions and logical (Boolean) operators, and returns a value of <code>TRUE</code> , <code>FALSE</code> , or <code>UNKNOWN</code> .
------------------	---

The `WHERE` clause can compare values in columns, literal, 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the WHERE Clause

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

Note: You cannot use column alias in the WHERE clause.

Character Strings and Dates

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

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

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-FEB-96' ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Character Strings and Dates

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

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

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.” Also, you learn about the use of single-row functions such as UPPER and LOWER to override the case sensitivity in the same lesson.

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Comparison Operators

Comparison operators are used in conditions that compare one expression with 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'
```

Remember, an alias cannot be used in the WHERE clause.

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

Using Comparison Operators

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Comparison Operators

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.

Range Conditions Using the BETWEEN Operator

Use the BETWEEN operator 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.

Range Conditions Using the BETWEEN Operator

You can display rows based on a range of values using the BETWEEN operator. 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 operator are inclusive. However, you must specify the lower limit first.

You can also use the BETWEEN operator 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

Membership Condition Using the IN Operator

Use the IN operator 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	Mourgos	5800	100
5	149	Zlotkey	10500	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Membership Condition Using the IN Operator

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

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

Note: The set of values can be specified in any random order—for example, (201,100,101).

The IN operator 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 with single quotation marks (' ').

Note: The IN operator is internally evaluated by the Oracle server as a set of OR conditions, such as a=value1 or a=value2 or a=value3. Therefore, using the IN operator has no performance benefits and is used only for logical simplicity.

Pattern Matching Using the LIKE Operator

- Use the LIKE operator 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%';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Pattern Matching Using the LIKE Operator

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

Symbol	Description
%	Represents any sequence of zero or more characters
_	Represents any single character

The SELECT statement in the slide returns the first name from the EMPLOYEES table for any employee whose first name begins with the letter “S.” Note the uppercase “S.” Consequently, names beginning with a lowercase “s” are not returned.

The LIKE operator 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';
```

Combining Wildcard Characters

- You can combine the two wildcard characters (% , _) with literal characters for pattern matching:

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

	LAST_NAME
1	Kochhar
2	Lorentz
3	Mourgos

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

ORACLE

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 Identifier

When you need to have an exact match for the actual % and _ characters, use the `ESCAPE` identifier. 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	149	Zlotkey	SA_MAN
2	174	Abel	SA_REP
3	176	Taylor	SA_REP
4	178	Grant	SA_REP

The `ESCAPE` identifier identifies the backslash (\) as the escape character. In the SQL statement, 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
1	King	(null)

ORACLE

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 that 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 example in the slide retrieves the last names and managers of all employees who do not have a manager.

Here is another example: To display the 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
1	Whalen	AD_ASST	(null)
2	Hartstein	MK_MAN	(null)
3	Fay	MK_REP	(null)
4	Higgins	AC_MGR	(null)
5	Gietz	AC_ACCOUNT	(null)

■ ■ ■

Defining Conditions Using the Logical Operators

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 condition is false

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Defining Conditions Using the Logical Operators

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 a single WHERE clause using the AND and OR operators.

Using the AND Operator

AND requires both the component 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the AND Operator

In the example, both the component conditions must be true for any record to be selected. Therefore, only those 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, that is, no rows are returned if 'MAN' is not uppercase. Further, character strings must be enclosed with 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 component 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the OR Operator

In the example, either component 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the NOT Operator

The example in the slide 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
```


Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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.

ORACLE

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 in the slide lists the default order of precedence. However, 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. Precedence of the AND Operator: Example

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. Using Parentheses: Example

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 the 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.”

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- **Sorting rows using the `ORDER BY` clause**
- Substitution variables
- `DEFINE` and `VERIFY` commands

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the ORDER BY Clause

- Sort the 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
5	Ernst	IT_PROG	60	21-MAY-91
6	De Haan	AD_VP	90	13-JAN-93

...

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. However, if you use the ORDER BY clause, it must be the last clause of the SQL statement. Further, 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.

Note: Use the keywords NULLS FIRST or NULLS LAST to specify whether returned rows containing null values should appear first or last in the ordering sequence.

Sorting


- Sorting in descending order:

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



- Sorting by column alias:

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Sorting

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 the alphabetical order (for example, “A” first and “Z” last).
- Null values are displayed last for ascending sequences and first for descending sequences.
- You can also sort by a column that is not in the SELECT list.

Examples:


1. To reverse the order in which the rows are displayed, specify the DESC keyword after the column name in the ORDER BY clause. The example in the slide sorts the result by the most recently hired employee.
2. You can also use a column alias in the ORDER BY clause. The slide example sorts the data by annual salary.

Note: The DESC keyword used here for sorting in descending order should not be confused with the DESC keyword used to describe table structures.

Sorting


- Sorting by using the column's numeric position:

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY 3;
```



- Sorting by multiple columns:

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Sorting (continued)

Examples:

3. You can sort query results by specifying the numeric position of the column in the SELECT clause. The example in the slide sorts the result by the `department_id` as this column is at the third position in the SELECT clause.
4. 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. The result of the query example shown in the slide is sorted by `department_id` in ascending order and also by salary in descending order.

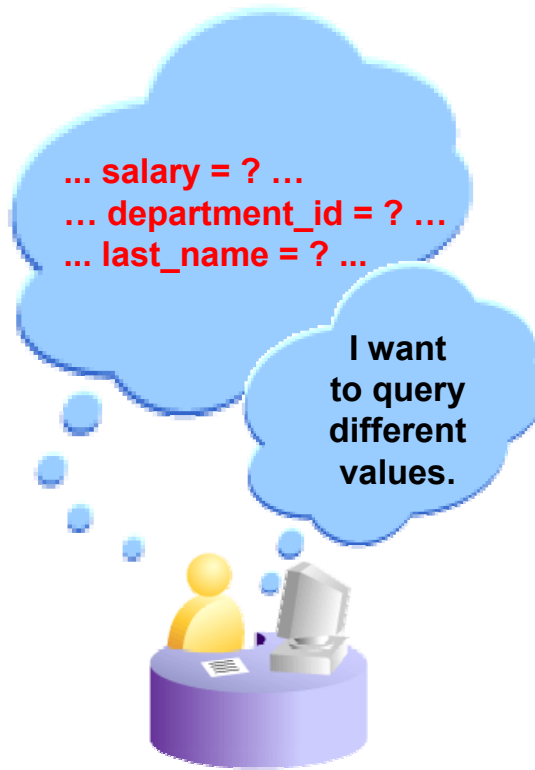
Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- **Substitution variables**
- `DEFINE` and `VERIFY` commands

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Substitution Variables



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Substitution Variables

So far, all the SQL statements were executed with predetermined columns, conditions, and their values. Suppose that you want a query that lists the employees with various jobs and not just those whose `job_id` is `SA_REP`. You can edit the `WHERE` clause to provide a different value each time you run the command, but there is also an easier way.

By using a substitution variable in place of the exact values in the `WHERE` clause, you can run the same query for different values.

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 values are temporarily stored. When the statement is run, the stored 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Substitution Variables (continued)

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

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

Restricted Ranges of Data: Examples

- 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 also be used to achieve other goals, such as:

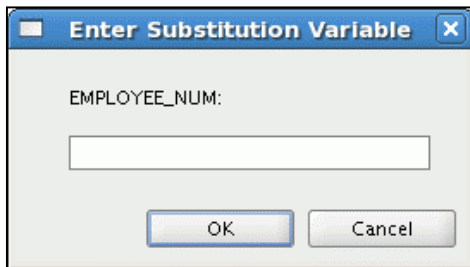
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

Note: Both SQL Developer and SQL* Plus support substitution variables and the DEFINE/UNDEFINE commands. Neither SQL Developer nor SQL* Plus support validation checks (except for data type) on user input. If used in scripts that are deployed to users, substitution variables can be subverted for SQL injection attacks.

Using the Single-Ampersand 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.

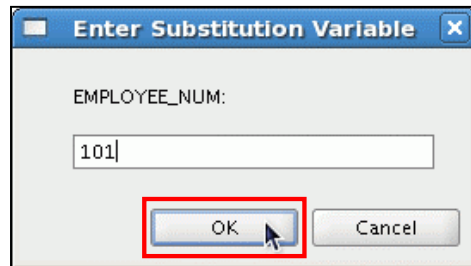
Using the Single-Ampersand Substitution Variable

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. However, you do not need to define the value of each variable.

Notation	Description
<i>&user_variable</i>	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 Single-Ampersand Substitution Variable

A dialog box titled "Enter Substitution Variable" with a close button (X) in the top right corner. Inside the dialog, the text "EMPLOYEE_NUM:" is followed by a text input field containing the value "101". Below the input field, there are two buttons: "OK" and "Cancel". The "OK" button is highlighted with a red rectangular border, and a mouse cursor is pointing at it.

Enter Substitution Variable

EMPLOYEE_NUM:

101

OK Cancel

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101 Kochhar	17000	90

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the Single-Ampersand Substitution Variable (continued)

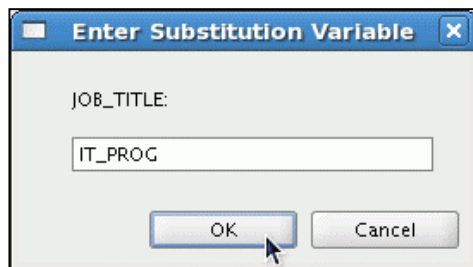
When SQL Developer detects that the SQL statement contains an ampersand, you are prompted to enter a value for the substitution variable that is named in the SQL statement.

After you enter a value and click the OK button, the results are displayed in the Results tab of your 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' ;
```

A dialog box titled "Enter Substitution Variable" with a close button (X). It contains a label "JOB_TITLE:" and a text input field containing "IT_PROG". At the bottom are "OK" and "Cancel" buttons. A mouse cursor is pointing at the "OK" button.

Enter Substitution Variable

JOB_TITLE:

IT_PROG

OK Cancel

	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.

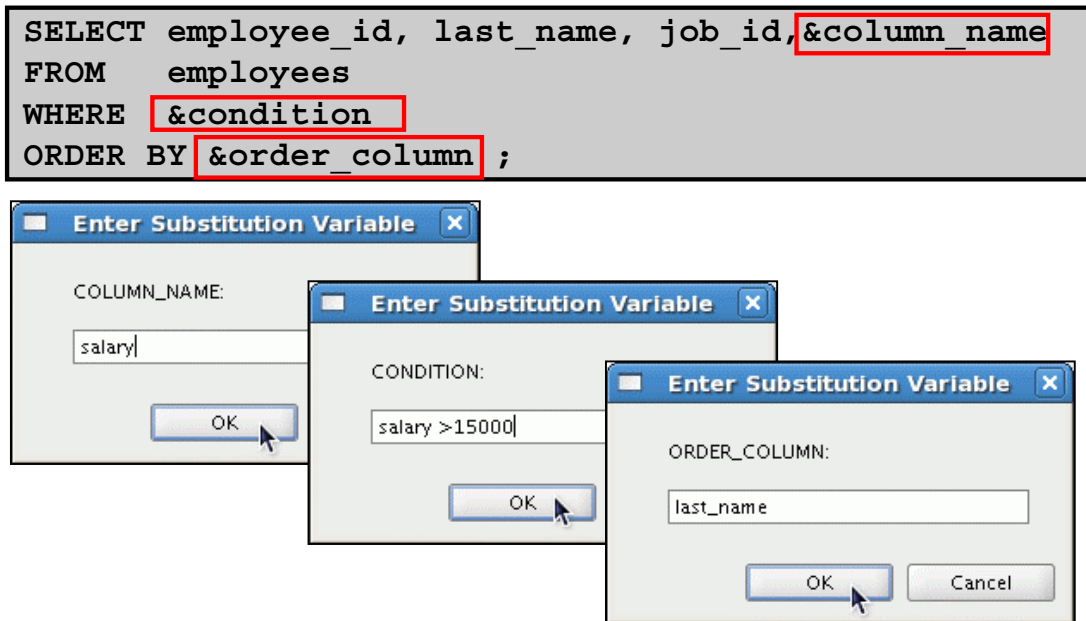
Character and Date Values with Substitution Variables

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

Enclose the variable with 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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Specifying Column Names, Expressions, and Text

You can use the substitution variables not only in the WHERE clause of a SQL statement, but also as substitution for column names, expressions, or text.

Example:

The example in the slide displays the employee number, last 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 then click OK to proceed.

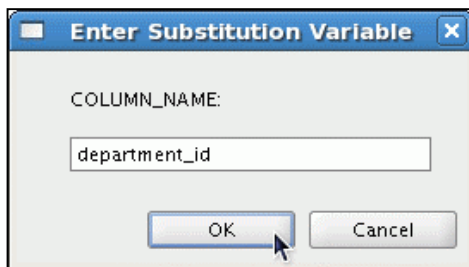
If you do not enter a value for the substitution variable, you get an error when you execute the preceding statement.

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

Using the Double-Ampersand Substitution Variable

Use 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 ;
```

A dialog box titled "Enter Substitution Variable" with a close button (X). It contains a label "COLUMN_NAME:" and a text input field with the value "department_id". Below the input field are "OK" and "Cancel" buttons. A mouse cursor is pointing at the "OK" button.

	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

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the 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 the variable, `column_name`, only once. The value that is supplied by the user (`department_id`) is used for both display and ordering of data. If you run the query again, you will not be prompted for the value of the variable.

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:

```
UNDEFINE column_name
```

Lesson Agenda

- Limiting rows with:
 - The `WHERE` clause
 - The comparison conditions using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` operators
 - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- Substitution variables
- **`DEFINE` and `VERIFY` commands**

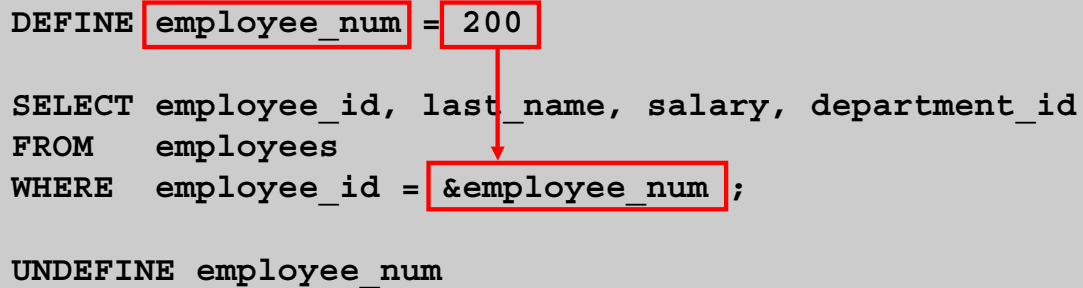
ORACLE

Copyright © 2009, Oracle. All rights reserved.

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
```

A red box highlights the variable 'employee_num' in the DEFINE statement and its value '200'. A red arrow points from the '200' to the '&employee_num' placeholder in the WHERE clause of the SQL query, illustrating the substitution process.

ORACLE

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.

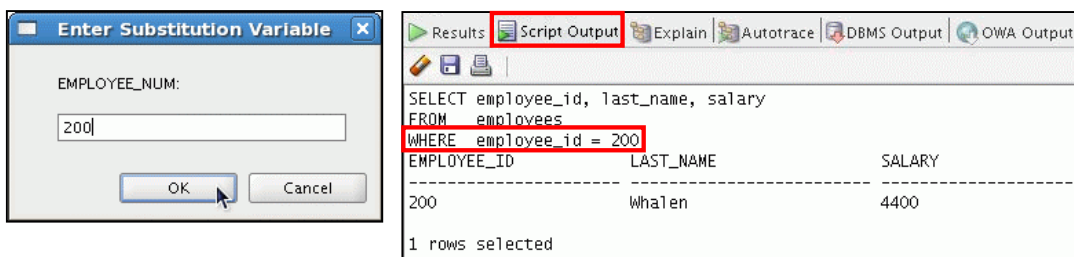
The EMPLOYEE_NUM substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

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:

```
SET VERIFY ON
```

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



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.

Quiz

Which of the following are valid operators for the WHERE clause?

1. >=
2. IS NULL
3. !=
4. IS LIKE
5. IN BETWEEN
6. <>

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answers: 1, 2, 3, 6

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` operators
 - 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

ORACLE

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. This enables the queries to prompt for the filter condition for the rows during run time.

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

ORACLE

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 the ampersand substitution.

3

Using Single-Row Functions to Customize Output

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

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

- Describe the various types of functions available in SQL
- Use the character, number, and date functions in `SELECT` statements

ORACLE

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.

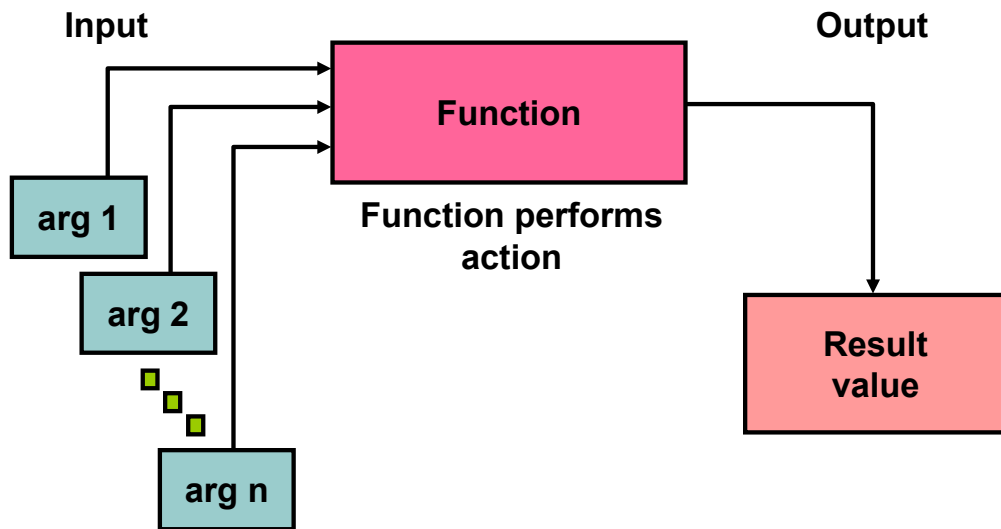
Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- Date functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Functions



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Functions

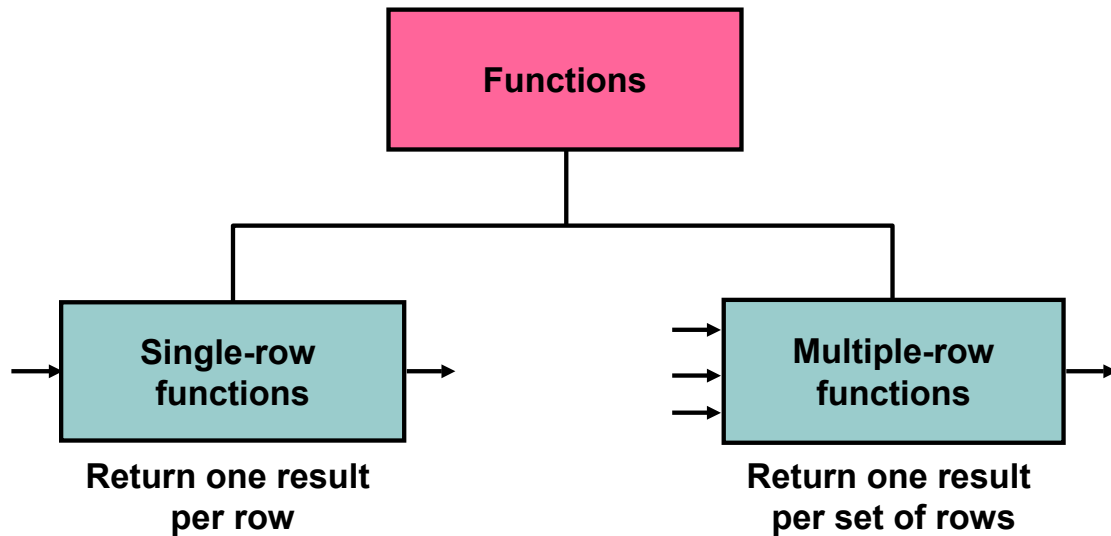
Functions are a very powerful feature of SQL. They can be used to do the following:

- 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: If you want to know whether a function is a SQL:2003 compliant function, refer to the *Oracle Compliance To Core SQL:2003* section in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Two Types of SQL Functions



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Two Types of SQL Functions

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 functions:

- 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 the lesson titled “Reporting Aggregated Data Using the Group Functions”).

Note: For more information and a complete list of available functions and their syntax, see the section on “Functions” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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,...)]
```

ORACLE

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

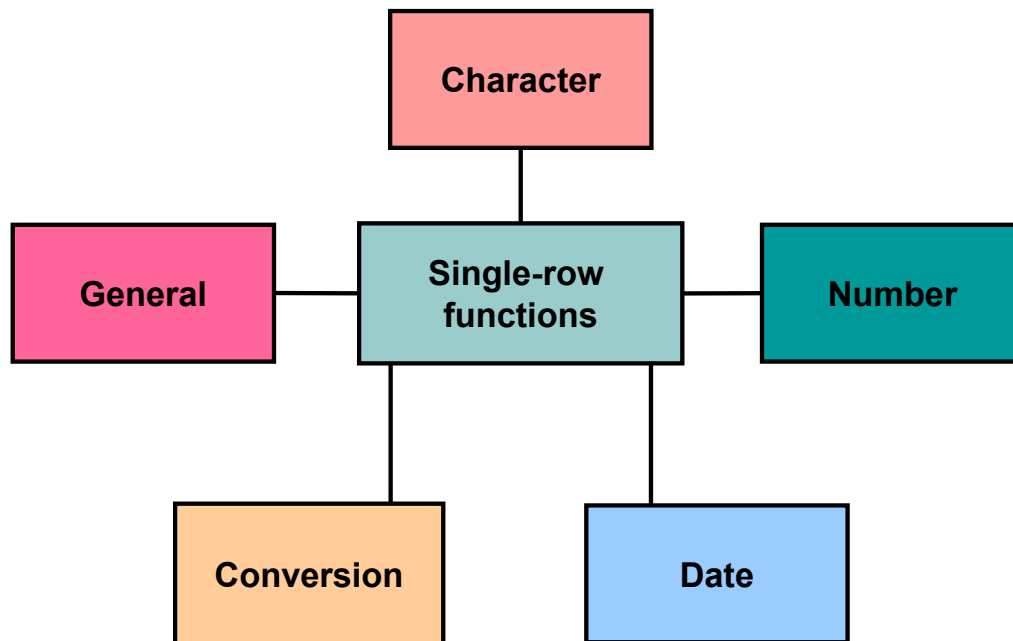
Features of single-row functions include:

- 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 the DATE data type except the MONTHS_BETWEEN function, which returns a number.)

The following single-row functions are discussed in the lesson titled “Using Conversion Functions and Conditional Expressions”:

- **Conversion functions:** Convert a value from one data type to another
- **General functions:**
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
 - CASE
 - DECODE

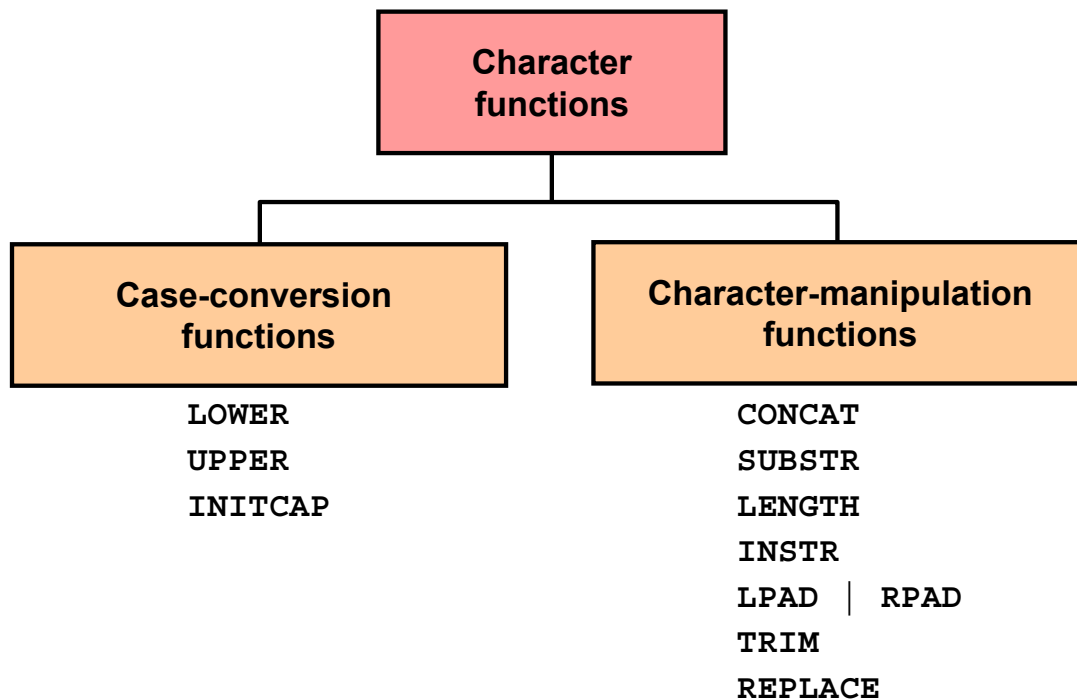
Lesson Agenda

- Single-row SQL functions
- **Character functions**
- Number functions
- Working with dates
- Date functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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-conversion functions
- Character-manipulation functions

Function	Purpose
LOWER(<i>column</i> / <i>expression</i>)	Converts alpha character values to lowercase
UPPER(<i>column</i> / <i>expression</i>)	Converts alpha character values to uppercase
INITCAP(<i>column</i> / <i>expression</i>)	Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase
CONCAT(<i>column1</i> / <i>expression1</i> , <i>column2</i> / <i>expression2</i>)	Concatenates the first character value to the second character value; equivalent to concatenation operator ()
SUBSTR(<i>column</i> / <i>expression</i> , <i>m</i> [<i>,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</i> / <i>expression</i>)	Returns the number of characters in the expression
INSTR(<i>column</i> / <i>expression</i> , ' <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 string and report the first occurrence.
LPAD(<i>column</i> <i>expression</i> , <i>n</i> , ' <i>string</i> ') RPAD(<i>column</i> <i>expression</i> , <i>n</i> , ' <i>string</i> ')	Returns an expression left-padded to length of <i>n</i> characters with a character expression. Returns an expression right-padded to length of <i>n</i> characters with a character expression.
TRIM(<i>leading</i> / <i>trailing</i> / <i>both</i> , <i>trim_character</i> FROM <i>trim_source</i>)	Enables you to trim leading 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. This is a feature that is available in Oracle8i and later versions.
REPLACE(<i>text</i> , <i>search_string</i> , <i>replacement_string</i>)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Note: Some of the functions that are fully or partially SQL:2003 compliant are:

UPPER

LOWER

TRIM

LENGTH

SUBSTR

INSTR

For more information, refer to the “Oracle Compliance To Core SQL:2003” section in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Case-Conversion Functions

These functions convert the case for character strings:

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Case-Conversion 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 the remaining letters to lowercase

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

A	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
4	The job id for ERNST is it_prog
5	The job id for FAY is mk_rep
6	The job id for GIETZ is ac_account

...

Using Case-Conversion 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';
```

0 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Case-Conversion 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 ('JACK and JUE', 'J', 'BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

ORACLE

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: Returns an expression left-padded to the length of *n* characters with a character expression
- RPAD: Returns an expression right-padded to the length of *n* characters with a character expression
- TRIM: Trims leading or trailing characters (or both) from a character string (If *trim_character* or *trim_source* is a character literal, you must enclose it within 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

The diagram illustrates the use of character-manipulation functions in an SQL query. The query is as follows:

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

Annotations 1, 2, and 3 point to specific parts of the query and the resulting table:

- Annotation 1 points to the `CONCAT(first_name, last_name)` expression in the `SELECT` clause.
- Annotation 2 points to the `LENGTH (last_name)` expression in the `SELECT` clause.
- Annotation 3 points to the `INSTR(last_name, 'a')` expression in the `SELECT` clause.

The resulting table shows the data for employees whose job ID contains 'REP' starting at the fourth position:

	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	KimberelyGrant	SA_REP	5	3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the Character-Manipulation Functions

The example in the slide 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

Lesson Agenda

- Single-row SQL functions
- Character functions
- **Number functions**
- Working with dates
- Date Functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Number Functions

- ROUND: Rounds value to a specified decimal
- TRUNC: Truncates value to a 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

ORACLE

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</i> <i>expression</i> , <i>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 the left of decimal point are rounded.)
TRUNC(<i>column</i> <i>expression</i> , <i>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</i> , <i>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 the section on “Numeric Functions” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Using the ROUND Function

The diagram illustrates the use of the ROUND function. It shows a SQL query and its output from the DUAL table. Annotations with numbered circles (1, 2, 3) and arrows point to specific parts of the query and results.

SQL Query:

```
SELECT ROUND(45.923, 2), ROUND(45.923, 0),  
       ROUND(45.923, -1)  
FROM   DUAL;
```

Results:

	ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
1	45.92	46	50

Annotations:

- Circle 1 points to the first argument (45.923) in the first ROUND function.
- Circle 2 points to the second argument (2) in the first ROUND function.
- Circle 3 points to the third argument (-1) in the third ROUND function.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the ROUND Function

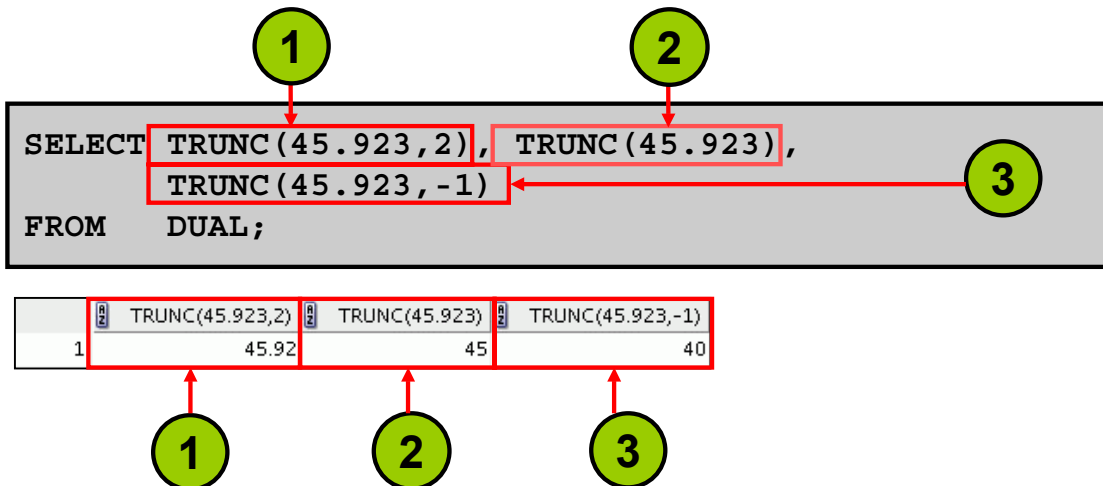
The ROUND function rounds the column, expression, or value to n decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left (rounded to the nearest unit of 100).

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 only once (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 completeness of the SELECT clause syntax, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from the actual tables.

Using the TRUNC Function



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the 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 the 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.

Using the 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 whether a value is odd or even. The MOD function is also the Oracle hash function.

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- **Working with dates**
- Date functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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.

Working with Dates

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 data might be June 17, 1987, 5:10:43 PM.

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

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
2048	27-OCT-52	1952	2052
2051	27-OCT-47	2147	2047

Note the values shown in the last two rows of the above table. As we approach the middle of the century, then the RR behavior is probably not what you want.

RR 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 uses 2 bytes for the year information, one for century and one for year. The century value is always included, whether or not it is specified or displayed. In this case, `RR` determines the default value for century on `INSERT`.

Using the SYSDATE Function

SYSDATE is a function that returns:

- Date
- Time

```
SELECT sysdate
FROM dual;
```

	SYSDATE
1	10-JUN-09

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the 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 public table called DUAL.

Note: SYSDATE returns the current date and time set for the operating system on which the database resides. Therefore, if you are in a place in Australia and connected to a remote database in a location in the United States (U.S.), the `sysdate` function will return the U.S. date and time. In that case, you can use the `CURRENT_DATE` function that returns the current date in the session time zone.

The `CURRENT_DATE` function and other related time zone functions are discussed in detail in the course titled *Oracle Database 11g: SQL Fundamentals II*.

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.

ORACLE

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
1	King	1147.102432208994708994708994708995
2	Kochhar	1028.959575066137566137566137566138
3	De Haan	856.102432208994708994708994708995

ORACLE

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 depending on the date and time set for the operating system of your local database when you run the SQL query.

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

Lesson Agenda

- Single-row SQL functions
- Character functions
- Number functions
- Working with dates
- **Date functions**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Date-Manipulation 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Date-Manipulation Functions

Date functions operate on Oracle dates. All date functions return a value of the 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*

The above list is a subset of the available date functions. ROUND and TRUNC number functions can also be used to manipulate the date values as shown below:

- 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.

The format models are covered in detail in the lesson titled “Using Conversion Functions and Conditional Expressions.”

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95' , '11-JAN-94')	19.6774194
ADD_MONTHS ('31-JAN-96' , 1)	'29-FEB-96'
NEXT_DAY ('01-SEP-95' , 'FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Date Functions

In the example in the slide, the ADD_MONTHS function adds one month to the supplied date value “31-JAN-96” and returns “29-FEB-96.” The function recognizes the year 1996 as the leap year and, therefore, returns the last day of the February month. If you change the input date value to “31-JAN-95,” the function returns “28-FEB-95.”

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and the last day of the hire month for all employees who have been employed for fewer than 150 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) < 150;
```

	EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DA...	LAST_DAY...
1	202	17-AUG-97	141.79757989...	17-FEB-98	22-AUG-97	31-AUG-97
2	107	07-FEB-99	124.12016054...	07-AUG-99	12-FEB-99	28-FEB-99
3	124	16-NOV-99	114.82983796...	16-MAY-00	19-NOV-99	30-NOV-99
4	142	29-JAN-97	148.41048312...	29-JUL-97	31-JAN-97	31-JAN-97
5	143	15-MAR-98	134.86209602...	15-SEP-98	20-MAR-98	31-MAR-98
6	144	09-JUL-98	131.05564441...	09-JAN-99	10-JUL-98	31-JUL-98
7	149	29-JAN-00	112.41048312...	29-JUL-00	04-FEB-00	31-JAN-00
8	176	24-MAR-98	134.57177344...	24-SEP-98	27-MAR-98	31-MAR-98
9	178	24-MAY-99	120.57177344...	24-NOV-99	28-MAY-99	31-MAY-99

Using ROUND and TRUNC Functions with Dates

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using ROUND and TRUNC Functions with Dates

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. If the format model is month, dates 1-15 result in the first day of the current month. Dates 16-31 result in the first day of the next month. If the format model is year, months 1-6 result in January 1 of the current year. Months 7-12 result in January 1 of the next year.

Example:

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and starting 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

Quiz

Which of the following statements are true about single-row functions?

1. Manipulate data items
2. Accept arguments and return one value per argument
3. Act on each row that is returned
4. Return one result per set of rows
5. May not modify the data type
6. Can be nested
7. Accept arguments that can be a column or an expression

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answers: 1, 3, 6, 7

Summary

In this lesson, you should have learned how to:

- Perform calculations on data using functions
- Modify individual data items using functions

ORACLE

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 values:** SYSDATE, MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY

Remember the following:

- Date values can also use arithmetic operators.
- ROUND and TRUNC functions can also be used with date values.

SYSDATE and DUAL

SYSDATE is a date function that returns the current date and time. It is customary to select SYSDATE from a single-row public table called DUAL.

Practice 3: Overview

This practice covers the following topics:

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 3: Overview

This practice provides a variety of exercises using different functions that are available for character, number, and date data types.

4

Using Conversion Functions and Conditional Expressions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

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

- Describe the various types of conversion functions that are available in SQL
- Use the `TO_CHAR`, `TO_NUMBER`, and `TO_DATE` conversion functions
- Apply conditional expressions in a `SELECT` statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

This lesson focuses on functions that convert data from one type to another (for example, conversion from character data to numeric data) and discusses the conditional expressions in SQL `SELECT` statements.

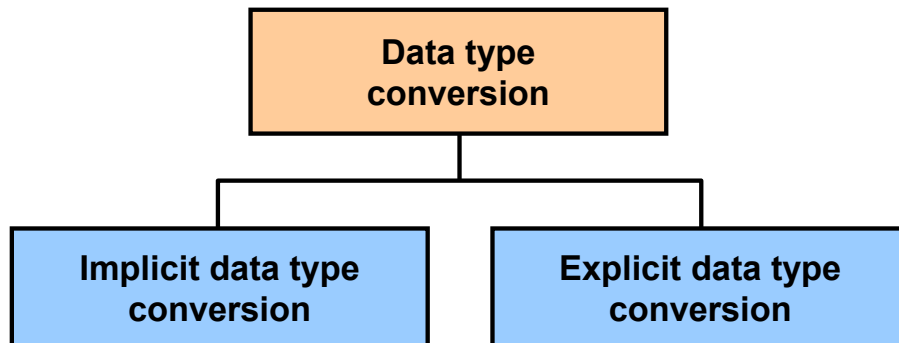
Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- Nesting functions
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - DECODE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Conversion Functions



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle Database can be defined by using the American National Standards Institute (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 receives 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 explained in the following slides.

Explicit data type conversions are performed 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 and the second data type is the output.

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

Implicit Data Type Conversion

In expressions, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Implicit Data Type Conversion

Oracle server can automatically perform data type conversion in an expression. For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string '01-JAN-90' to a date. Therefore, a VARCHAR2 or CHAR value can be implicitly converted to a number or date data type in an expression.

Implicit Data Type Conversion

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

From	To
NUMBER	VARCHAR2 or CHAR
DATE	VARCHAR2 or CHAR

ORACLE

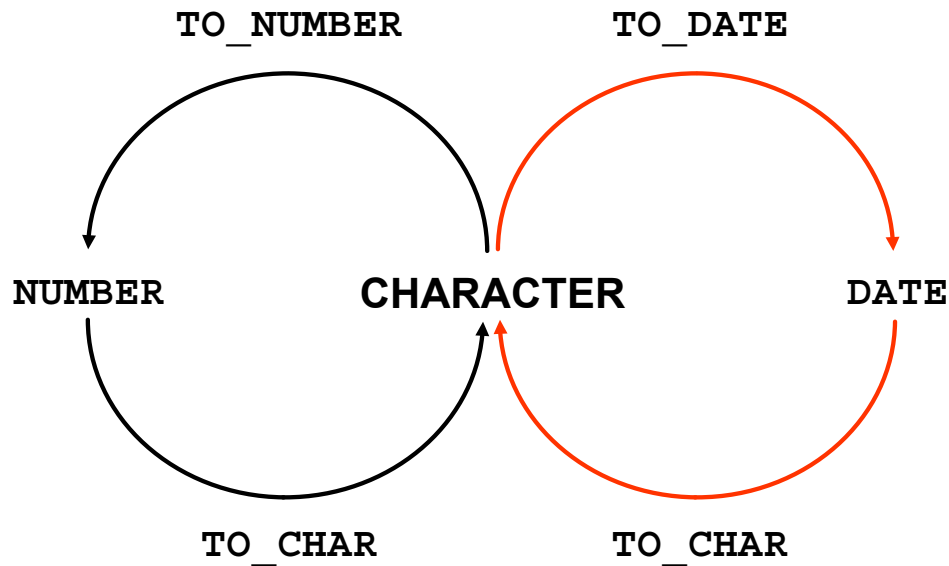
Copyright © 2009, Oracle. All rights reserved.

Implicit Data Type Conversion (continued)

In general, the Oracle server uses the rule for expressions when a data type conversion is needed. For example, the expression `grade = 2` results in the implicit conversion of the number 2 to the string “2” because `grade` is a `CHAR(2)` column.

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

Explicit Data Type Conversion



ORACLE

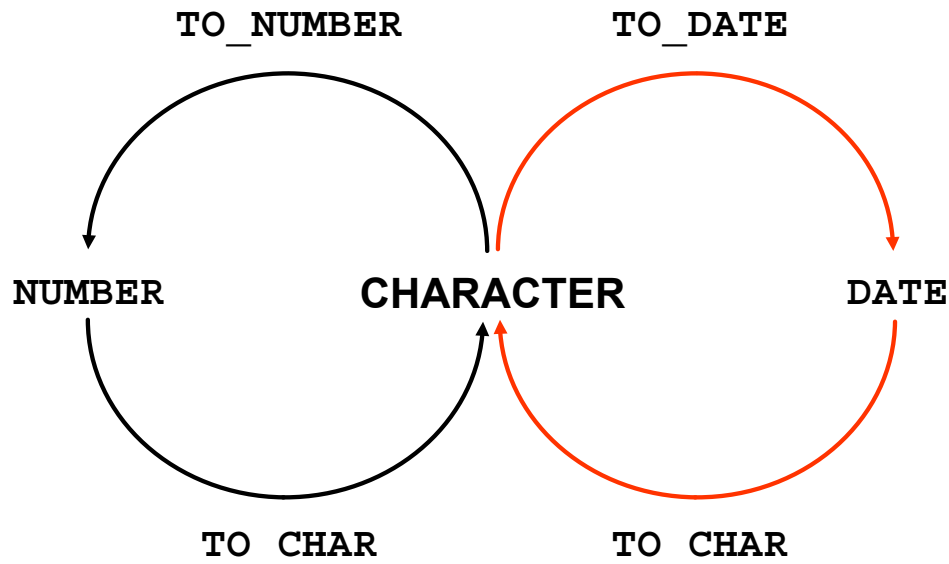
Copyright © 2009, Oracle. All rights reserved.

Explicit Data Type Conversion

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

Function	Purpose
<code>TO_CHAR(<i>number</i> <i>date</i>, [<i>fmt</i>], [<i>nlsparams</i>])</code>	<p>Converts a number or date value to a VARCHAR2 character string with the format model <i>fmt</i></p> <p>Number conversion: The <i>nlsparams</i> parameter specifies the following characters, which are returned by number format elements:</p> <ul style="list-style-type: none">• Decimal character• Group separator• Local currency symbol• International currency symbol <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>	Date conversion: The <code>nlsparams</code> parameter specifies the language in which the 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 <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 the section on “Conversion Functions” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Implicit and explicit data type conversion
- **TO_CHAR, TO_DATE, TO_NUMBER functions**
- Nesting functions
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - DECODE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the TO_CHAR Function with Dates

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

The format model:

- Must be enclosed with 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the TO_CHAR Function with Dates

TO_CHAR converts a datetime data type to a value of VARCHAR2 data type in the format specified by the *format_model*. A format model is a character literal that describes the format of datetime stored in a character string. For example, the datetime format model for the string '11-Nov-1999' is 'DD-Mon-YYYY'. You can use the TO_CHAR function to convert a date from its default format to the one that you specify.

Guidelines

- The format model must be enclosed with single quotation marks and is case-sensitive.
- The format model can include any valid date format element. But be sure to separate the date value from the format model with 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 the 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 digit of the 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 the month padded with blanks to a length of nine characters
MON	Name of the month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of the year or month
DDD or DD or D	Day of the year, month, or week
DAY	Name of the day padded with blanks to a length of nine characters
DY	Name of the day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.
IW	Weeks in the year from ISO standard (1 to 53)

Elements of the Date Format Model

- Time elements format the time portion of the date:

HH24:MI:SS AM	15:45:32 PM
---------------	-------------

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

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- Number suffixes spell out numbers:

ddspth	fourteenth
--------	------------

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Elements of the Date Format Model

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)
SSSSS	Seconds past midnight (0–86399)

Elements of the Date Format Model (continued)

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, DDSP 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
6	King	17 June 1987
7	Kochhar	21 September 1989
8	De Haan	13 January 1993
9	Hunold	3 January 1990
10	Ernst	21 May 1991

...

ORACLE

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 example in the slide 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

...

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

ORACLE

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	0999999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns the decimal character in the specified position. 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	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
U	Returns in the specified position the “Euro” (or other) dual currency	U9999	€1234
V	Multiply by 10 <i>n</i> times (<i>n</i> = number of 9s after V)	9999V99	123400
S	Returns the negative or positive value	S9999	-1234 or +1234
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';
```

	1	SALARY
	1	\$6,000.00

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the TO_CHAR Function with Numbers (continued)

- The Oracle server displays a string of number signs (#) in place of a whole number whose digits exceed the number of digits provided in the format model.
- The Oracle server rounds the stored decimal value to the number of decimal places 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 match for the character argument and date format model of a TO_DATE function.

ORACLE

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 select is based on the previously demonstrated format elements.

The `fx` modifier specifies the exact match 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`, the Oracle server 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`, the numbers in the character argument can omit leading zeros.

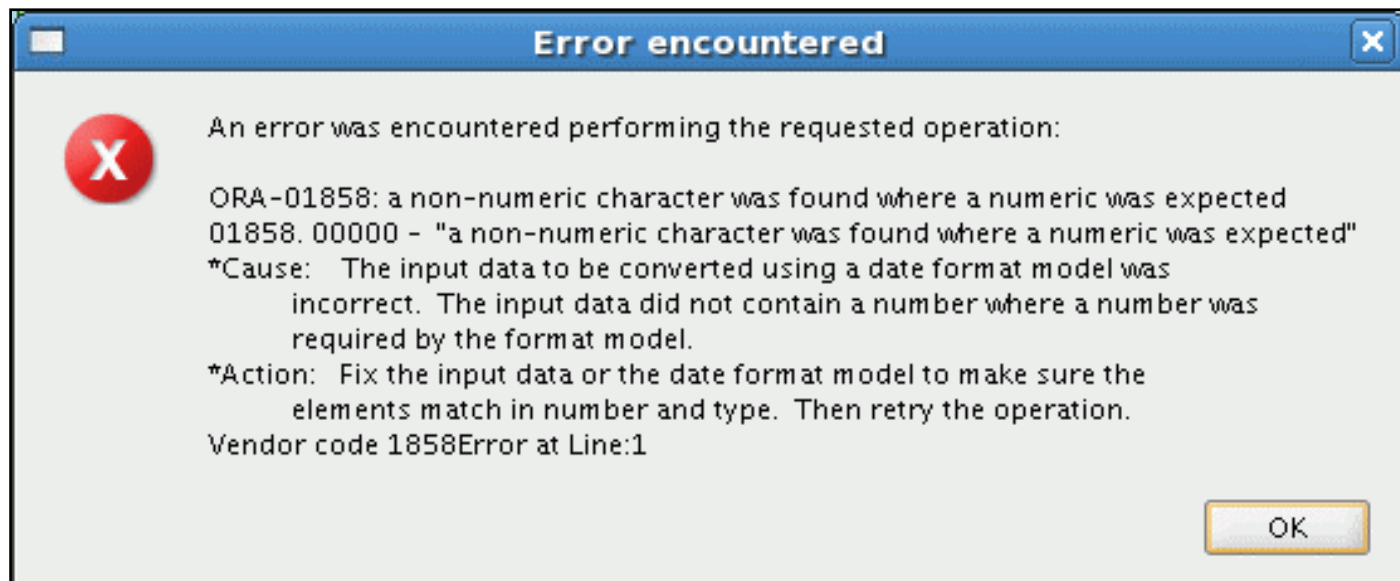
Using the TO_NUMBER and TO_DATE Functions (continued)

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 before 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');
```

The resulting error output looks like this:



To see the output, correct the query by deleting the extra space between 'May' and '24'.

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

	LAST_NAME	HIRE_DATE
1	Grant	24-MAY-99

Using the TO_CHAR and TO_DATE Function with the RR Date Format

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the TO_CHAR and TO_DATE Function with the RR Date Format

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.

Alternatively, the following command, 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';
```

```
0 rows selected
```

Lesson Agenda

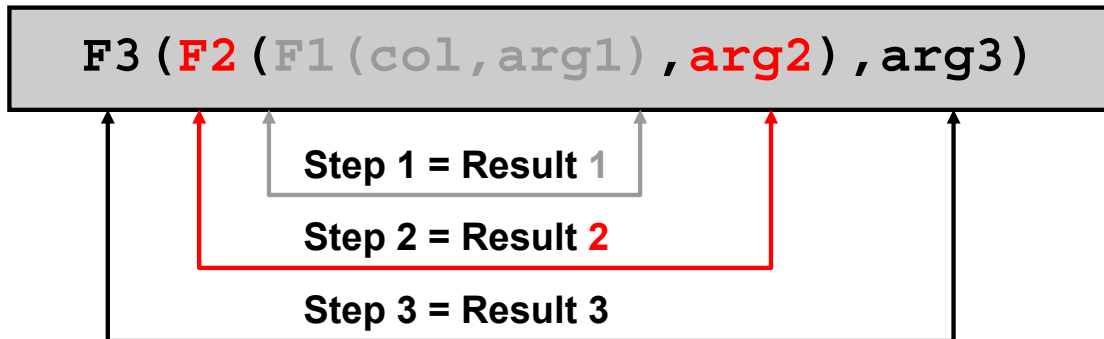
- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- **Nesting functions**
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - DECODE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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: Example 1

```
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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Nesting Functions (continued)

The example in the slide 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;
```

Nesting Functions: Example 2

```
SELECT      TO_CHAR(ROUND((salary/7), 2), '99G999D99',  
              'NLS_NUMERIC_CHARACTERS = ','.' ')  
              "Formatted Salary"  
FROM employees;
```

	Formatted Salary
1	628,57
2	1.857,14
3	857,14
4	1.714,29
5	1.185,71
6	3.428,57

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Nesting Functions (continued)

The example in the slide displays the salaries of employees divided by 7 and rounded to two decimals. The result is then formatted to display the salary in Danish notation. That is, comma is used for decimal point and a period for thousands.

First, the inner ROUND function is executed to round off the value of salary divided by 7 to two decimal places. The TO_CHAR function is then used to format the result of the ROUND function.

Note: D and G specified in the TO_CHAR function parameter are number format elements. D returns a decimal character in the specified position. G is used as a group separator.

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- Nesting functions
- **General functions:**
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- **Conditional expressions:**
 - CASE
 - DECODE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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)

ORACLE

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 <code>expr1</code> is not null, NVL2 returns <code>expr2</code> . If <code>expr1</code> is null, NVL2 returns <code>expr3</code> . The argument <code>expr1</code> 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 the section on “Functions” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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')`

ORACLE

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(number_column, 9)</code>
DATE	<code>NVL(date_column, '01-JAN-95')</code>
CHAR or VARCHAR2	<code>NVL(character_column, '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
1	Whalen	4400	0	52800
2	Hartstein	13000	0	156000
3	Fay	6000	0	72000
4	Higgins	12000	0	144000
5	Gietz	8300	0	99600
6	King	24000	0	288000
7	Kochhar	17000	0	204000
8	De Haan	17000	0	204000
9	Hunold	9000	0	108000
10	Ernst	6000	0	72000

...

1

2

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
1	Whalen	4400	(null)	(null)

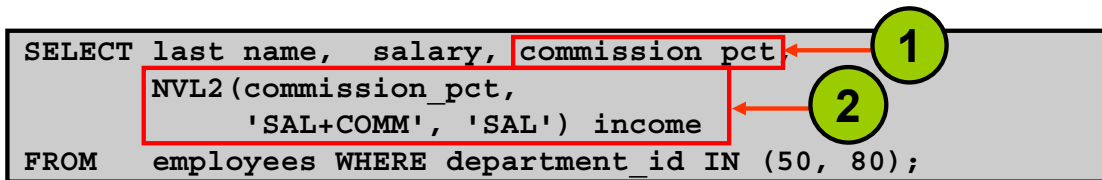
...

16	Vargas	2500	(null)	(null)
17	Zlotkey	10500	0.2	151200
18	Abel	11000	0.3	171600
19	Taylor	8600	0.2	123840
20	Grant	7000	0.15	96600

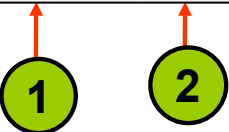
Notice that the annual compensation is calculated for only those employees who earn a commission. If any column value in an expression is null, the result is null. To calculate values for all employees, you must convert the null value to a number before applying the arithmetic operator. In the example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

Using the NVL2 Function

```
SELECT last name, salary, commission_pct,
       NVL2 (commission_pct,
             'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```



	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	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, the NVL2 function returns the second expression. If the first expression is null, the third expression is returned.

Syntax

`NVL2(expr1, expr2, expr3)`

In the syntax:

- *expr1* is the source value or expression that may contain a 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 text literal value of SAL+COMM is returned. If the COMMISSION_PCT column contains a null value, the text literal value of SAL is returned.

Note: The argument *expr1* can have any data type. The arguments *expr2* and *expr3* can have any data types except LONG.

Using the NULLIF Function

```
SELECT first_name, LENGTH(first_name) "expr1",  
       last_name, LENGTH(last_name) "expr2",  
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result  
FROM employees;
```

	FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
1	Ellen	5	Abel	4	5
2	Curtis	6	Davies	6	(null)
3	Lex	3	De Haan	7	3
4	Bruce	5	Ernst	5	(null)
5	Pat	3	Fay	3	(null)
6	William	7	Gietz	5	7
7	Kimberely	9	Grant	5	9
8	Michael	7	Hartstein	9	7
9	Shelley	7	Higgins	7	(null)
...					

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the NULLIF Function

The NULLIF function compares two expressions.

Syntax

```
NULLIF (expr1, expr2)
```

In the syntax:

- NULLIF compares *expr1* and *expr2*. If they are equal, the function returns null. If they are not, the function returns *expr1*. However, you cannot specify the literal NULL for *expr1*.

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.

ORACLE

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

Note that all expressions must be of the same data type.

Using the COALESCE Function

```
SELECT last name, employee id,  
COALESCE(TO_CHAR(commission_pct), TO_CHAR(manager_id),  
         'No commission and no manager')  
FROM employees;
```

	LAST_NAME	EMPLOYEE_ID	COALESCE(TO_CHAR(COMMISSION_PCT), TO_CHAR(MANAGER_ID), 'No commission and no manager')
1	Whalen	200	101
2	Hartstein	201	100
3	Fay	202	201
4	Higgins	205	101
5	Gietz	206	205
6	King	100	No commission and no manager
...			
17	Zlotkey	149	.2
18	Abel	174	.3
19	Taylor	176	.2
20	Grant	178	.15

ORACLE

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, the `commission_pct` is displayed. If the `manager_id` and `commission_pct` values are null, “No commission and no manager” is displayed. Note that `TO_CHAR` function is applied so that all expressions are of the same data type.

Using the COALESCE Function (continued)

Example:

For the employees who do not get any commission, your organization wants to give a salary increment of \$2,000 and for employees who get commission, the query should compute the new salary that is equal to the existing salary added to the commission amount.

```
SELECT last_name, salary, commission_pct,  
       COALESCE((salary+(commission_pct*salary)), salary+2000, salary) "New  
       Salary"  
FROM   employees;
```

Note: Examine the output. For employees who do not get any commission, the New Salary column shows the salary incremented by \$2,000 and for employees who get commission, the New Salary column shows the computed commission amount added to the salary.

	A Z LAST_NAME	A Z SALARY	A Z COMMISSION_PCT	A Z NewSalary
1	Whalen	4400	(null)	6400
2	Hartstein	13000	(null)	15000
3	Fay	6000	(null)	8000
4	Higgins	12000	(null)	14000
5	Gietz	8300	(null)	10300
6	King	24000	(null)	26000

■ ■ ■

17	Zlotkey	10500	0.2	12600
18	Abel	11000	0.3	14300
19	Taylor	8600	0.2	10320
20	Grant	7000	0.15	8050

Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- Nesting functions
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - DECODE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Conditional Expressions

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Conditional Expressions

The two methods that are 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 the 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
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CASE Expression

CASE expressions allow you to use the 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, the Oracle server returns *else_expr*. Otherwise, the Oracle server returns a null. You cannot specify the literal NULL for all the *return_exprs* and the *else_expr*.

The expressions *expr* and *comparison_expr* must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2. All of the return values (*return_expr*) must be of the same data type.

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
1	Whalen	AD_ASST	4400	4400
...				
9	Hunold	IT_PROG	9000	9900
10	Ernst	IT_PROG	6000	6600
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
14	Davies	ST_CLERK	3100	3565
...				
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

ORACLE

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 code is an example of the 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, a 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])
```

ORACLE

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
...				
10	Ernst	IT_PROG	6000	6600
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
...				
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;
```

ORACLE

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%

R2	LAST_NAME	R2	SALARY	R2	TAX_RATE
1	Zlotkey		10500		0.42
2	Abel		11000		0.42
3	Taylor		8600		0.4

Quiz

The `TO_NUMBER` function converts either character strings or date values to a number in the format specified by the optional format model.

1. True
2. False

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answer: 2

Summary

In this lesson, you should have learned how to:

- Alter date formats for display using functions
- Convert column data types using functions
- Use NVL functions
- Use IF-THEN-ELSE logic and other conditional expressions in a SELECT statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

Remember the following:

- 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.
- The IF-THEN-ELSE logic can be applied within a SQL statement by using the CASE expression or the DECODE function.

Practice 4: Overview

This practice covers the following topics:

- Creating queries that use TO_CHAR, TO_DATE, and other DATE functions
- Creating queries that use conditional expressions such as DECODE and CASE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 4: Overview

This practice provides a variety of exercises using TO_CHAR and TO_DATE functions, and conditional expressions such as DECODE and CASE. Remember that for nested functions, the results are evaluated from the innermost function to the outermost function.

5

Reporting Aggregated Data Using the Group Functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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

ORACLE

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.

Lesson Agenda

- Group functions:
 - Types and syntax
 - Use AVG, SUM, MIN, MAX, COUNT
 - Use the DISTINCT keyword within group functions
 - NULL values in a group function
- Grouping rows:
 - GROUP BY clause
 - HAVING clause
- Nesting group functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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
...		
18	80	11000
19	80	8600
20	(null)	7000

Maximum salary in
EMPLOYEES table

MAX(SALARY)
24000

ORACLE

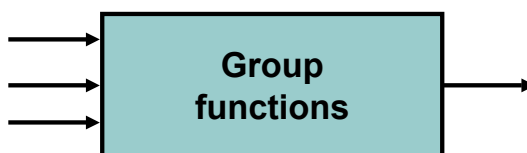
Copyright © 2009, Oracle. All rights reserved.

What Are 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 <u>ALL</u>] <i>n</i>)	Average value of <i>n</i> , ignoring null values
COUNT ({ * [DISTINCT <u>ALL</u>] <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 <u>ALL</u>] <i>expr</i>)	Maximum value of <i>expr</i> , ignoring null values
MIN ([DISTINCT <u>ALL</u>] <i>expr</i>)	Minimum value of <i>expr</i> , ignoring null values
STDDEV ([DISTINCT <u>ALL</u>] <i>n</i>)	Standard deviation of <i>n</i> , ignoring null values
SUM ([DISTINCT <u>ALL</u>] <i>n</i>)	Sum values of <i>n</i> , ignoring null values
VARIANCE ([DISTINCT <u>ALL</u>] <i>n</i>)	Variance of <i>n</i> , ignoring null values

Group Functions: Syntax

```
SELECT    group_function(column), ...
FROM      table
[WHERE    condition]
[ORDER BY column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Group Functions: Syntax

The group function is placed after the `SELECT` keyword. You may have multiple group functions separated by commas.

Guidelines for using the 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`, `COALESCE`, `CASE`, or `DECODE` 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 AVG and SUM Functions

You can use the AVG, SUM, MIN, and MAX functions against the 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the MIN and MAX Functions

You can use the MAX and MIN functions for numeric, character, and date data types. The example in the slide 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 alphabetic list of all employees:

```
SELECT MIN(last_name), MAX(last_name)
FROM   employees;
```

	MIN(LAST_NAME)	MAX(LAST_NAME)
1	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.

Using the 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 example in the slide displays the number of employees in department 50.
2. The example in the slide 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.

Using the 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.

However, the NVL function forces group functions to include null values.

Examples:

1. The average is calculated based on *only* those rows in the table in which 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).

Lesson Agenda

- Group functions:
 - Types and syntax
 - Use AVG, SUM, MIN, MAX, COUNT
 - Use DISTINCT keyword within group functions
 - NULL values in a group function
- Grouping rows:
 - GROUP BY clause
 - HAVING clause
- Nesting group functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Groups of Data

EMPLOYEES

R	DEPARTMENT_ID	R	SALARY	
1	10		4400	4400
2	20		13000	
3	20		6000	9500
4	50		2500	
5	50		2600	
6	50		3100	3500
7	50		3500	
8	50		5800	
9	60		9000	
10	60		6000	6400
11	60		4200	
12	80		11000	
13	80		8600	10033
...				
18	110		8300	
19	110		12000	
20	(null)		7000	

**Average salary in the
EMPLOYEES table for
each department**

R	DEPARTMENT_ID	R	AVG(SALARY)
1	(null)		7000
2	20		9500
3	90		19333.333333333333...
4	110		10150
5	50		3500
6	80		10033.333333333333...
7	10		4400
8	60		6400

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Groups of Data

Until this point in the 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. This can be done by using the GROUP BY clause.

Creating Groups of Data: GROUP BY Clause Syntax

You can divide rows in a table into smaller groups by using the GROUP BY clause.

```
SELECT    column, group_function(column)
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Groups of Data: GROUP BY Clause Syntax

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 the 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 the 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	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

ORACLE

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 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.

Note: To order the query results in ascending or descending order, include the ORDER BY clause in the query.

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 ;
```

	A	AVG(SALARY)
1		7000
2		9500
3		19333.33333333333333333333...
4		10150
5		3500
6		10033.33333333333333333333...
7		4400
8		6400

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 also use the group function in the ORDER BY clause:

```
SELECT  department_id, AVG(salary)
FROM    employees
GROUP BY department_id
ORDER BY AVG(salary) ;
```

	A	DEPARTMENT_ID	A	AVG(SALARY)
1		50		3500
2		10		4400
3		60		6400

...

7		110		10150
8		90		19333.33333333333333333333...

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
...			
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	110	AC_ACCOUNT	8300
2	110	AC_MGR	12000
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Grouping by More Than One Column

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 the department number, and then by the 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.

The following **SELECT** statement returns the result shown in the slide:

```
SELECT  department_id, job_id, sum(salary)
FROM    employees
GROUP BY department_id, job_id
ORDER BY job_id;
```

Using the GROUP BY Clause on Multiple Columns

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE     department_id > 40
GROUP BY  department_id, job_id
ORDER BY  department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the Group By Clause on Multiple Columns

You can return summary results for groups and subgroups by listing multiple GROUP BY columns. The GROUP BY clause groups rows but does not guarantee the order of the result set. To order the groupings, use the ORDER BY clause.

In the example in the slide, the SELECT statement that contains a GROUP BY clause is evaluated as follows:

- The SELECT clause specifies the column to be retrieved:
 - Department ID in the EMPLOYEES table
 - Job ID in the EMPLOYEES table
 - The sum of all 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 reduces the result set to those rows where department ID is greater than 40.
- The GROUP BY clause specifies how you must group the resulting rows:
 - First, the rows are grouped by the department ID.
 - Second, the rows are grouped by job ID in the department ID groups.
- The ORDER BY clause sorts the results by department ID.

Note: The SUM function is applied to the salary column for all job IDs in the result set in each department ID group. Also, note that the SA_REP row is not returned. The department ID for this row is NULL and, therefore, does not meet the WHERE condition.

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;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"

A GROUP BY clause must be added to count the last names for each department_id.

```
SELECT department_id, job_id, COUNT(last_name)
FROM employees
GROUP BY department_id;
```

ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"

Either add job_id in the GROUP BY or remove the job_id column from the SELECT list.

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, the error message “not a single-group group function” appears and an asterisk (*) points to the offending column. You can correct the error in the first example in the slide by adding the GROUP BY clause:

```
SELECT department_id, count(last_name)
FROM employees
GROUP BY department_id;
```

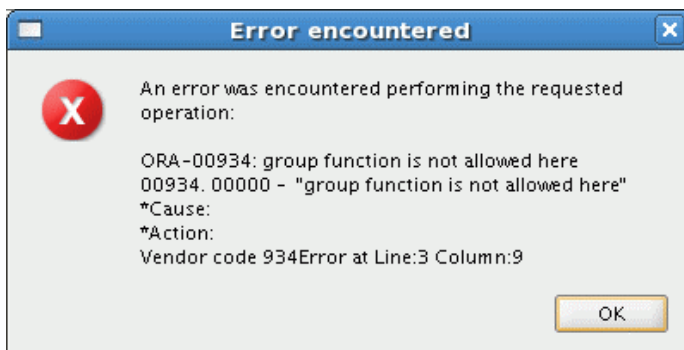
Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause. In the second example in the slide, job_id is neither in the GROUP BY clause nor is it being used by a group function, so there is a “not a GROUP BY expression” error. You can correct the error in the second slide example by adding job_id in the GROUP BY clause.

```
SELECT department_id, job_id, COUNT(last_name)
FROM employees
GROUP BY department_id, job_id;
```

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 example in the slide results in an error because it uses the `WHERE` clause to restrict the display of the average salaries of those departments that have an average salary greater than \$8,000.

However, you can correct the error in the example by using the `HAVING` clause to restrict groups:

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id
HAVING    AVG(salary) > 8000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	20	9500
2	90	19333.3333333333...
3	110	10150
4	80	10033.3333333333...

Restricting Group Results

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
...		
18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than \$10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Restricting Group Results

You use the **HAVING** clause to restrict groups in the same way that you use the **WHERE** clause to restrict the rows that you select. 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:

1. Rows are grouped.
2. The group function is applied.
3. 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 the groups that 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 it is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Note: The WHERE clause restricts rows, whereas the HAVING clause restricts groups.

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 example in the slide displays the department numbers and maximum salaries for those departments with a maximum salary 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 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	19333.333333333...
3	110	10150
4	80	10033.333333333...

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);
```

	<small>A Z</small> JOB_ID	<small>A Z</small> PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the HAVING Clause (continued)

The example in the slide 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.

Lesson Agenda

- Group functions:
 - Types and syntax
 - Use AVG, SUM, MIN, MAX, COUNT
 - Use DISTINCT keyword within group functions
 - NULL values in a group function
- Grouping rows:
 - GROUP BY clause
 - HAVING clause
- Nesting group functions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Nesting Group Functions

Display the maximum average salary:

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

[illegible]

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Nesting Group Functions

Group functions can be nested to a depth of two functions. The example in the slide calculates the average salary for each department `id` and then displays the maximum average salary.

Note that GROUP BY clause is mandatory when nesting group functions.

Quiz

Identify the guidelines for group functions and the GROUP BY clause.

1. You cannot use a column alias in the GROUP BY clause.
2. The GROUP BY column must be in the SELECT clause.
3. By using a WHERE clause, you can exclude rows before dividing them into groups.
4. The GROUP BY clause groups rows and ensures order of the result set.
5. If you include a group function in a SELECT clause, you cannot select individual results as well.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answers: 1, 3

Summary

In this lesson, you should have learned how to:

- Use the group functions COUNT, MAX, MIN, SUM, 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];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

There are several group functions available in SQL, such as AVG, COUNT, MAX, MIN, SUM, STDDEV, and VARIANCE.

You can create subgroups by using the GROUP BY clause. Further, 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 GROUP BY and HAVING clauses following the WHERE clause is not important. Place the ORDER BY clause at the end.

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 Database SQL Language Reference 11g, Release 1 (11.1)*.

Practice 5: Overview

This practice covers the following topics:

- Writing queries that use the group functions
- Grouping by rows to achieve more than one result
- Restricting groups by using the `HAVING` clause

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 5: Overview

In this practice, you learn to use group functions and select groups of data.

6

Displaying Data from Multiple Tables Using Joins

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

ORACLE

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 about joins is found in the section on “SQL Queries and Subqueries: Joins” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
...			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS

	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

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping

...			
18	205	110	Accounting
19	206	110	Accounting

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 example in the slide, 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:

- Natural joins:
 - NATURAL JOIN clause
 - USING clause
 - ON clause
- OUTER joins:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cross joins

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Types of Joins

To join tables, you can use a join syntax that is compliant with the SQL:1999 standard.

Note

- Before the Oracle9i release, the join syntax was different from the American National Standards Institute (ANSI) standards. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in the prior releases. For detailed information about the proprietary join syntax, see Appendix F: Oracle Join Syntax.
- The following slide discusses the SQL:1999 join syntax.

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];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Joining Tables Using SQL:1999 Syntax

In the syntax:

- `table1.column` denotes the table and the column from which data is retrieved
- `NATURAL JOIN` joins two tables based on the same column name
- `JOIN table2 USING column_name` performs an equijoin based on the column name
- `JOIN table2 ON table1.column_name = table2.column_name` performs an equijoin based on the condition in the `ON` clause
- `LEFT/RIGHT/FULL OUTER` is used to perform OUTER joins
- `CROSS JOIN` returns a Cartesian product from the two tables

For more information, see the section titled “SELECT” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
 - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Qualifying Ambiguous Column Names

When joining two or more tables, 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. 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.

However, qualifying column names with table names can be time consuming, particularly if the table names are lengthy. Instead, you can use *table aliases*. 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.

The table name is specified in full, followed by a space, and then the table alias. For example, the `EMPLOYEES` table can be 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, 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.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all the 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 the columns in the two tables that have matching data types and names. You do this by using the `NATURAL JOIN` keywords.

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, 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, use the `USING` clause to specify the columns for the equijoin.
- Use the `USING` clause to match only one column when more than one column matches.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Joins with the 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.

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
...		

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Foreign key

Primary key

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Joining Column Names

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 the 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 employee_id, last_name,  
       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	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50

...

18	206	Gietz	1700	110
19	205	Higgins	1700	110

ORACLE

Copyright © 2009, Oracle. All rights reserved.

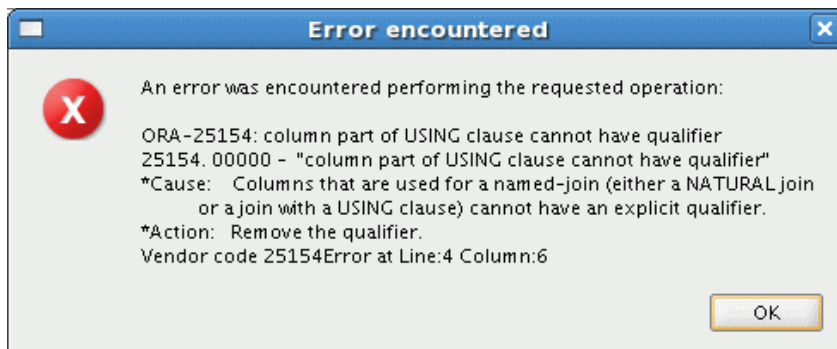
Retrieving Records with the USING Clause

In the example in the slide, the DEPARTMENT_ID columns in the EMPLOYEES and DEPARTMENTS tables are joined and thus the LOCATION_ID of the department where an employee works is shown.

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the USING clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE  d.location_id = 1400;
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Table Aliases with the USING clause

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. For example, in the query mentioned in the slide, you should not alias the `location_id` column in the WHERE clause because the column is used in the USING clause.

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 columns that are common in both the tables, but not used in the USING clause, must be prefixed with a table alias; otherwise, you get the “column ambiguously defined” error.

In the following statement, `manager_id` is present in both the `employees` and `departments` table; if `manager_id` is not prefixed with a table alias, it gives a “column ambiguously defined” error.

The following statement is valid:

```
SELECT first_name, d.department_name, d.manager_id
FROM   employees e JOIN departments d USING (department_id)
WHERE  department_id = 50;
```

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.

Creating Joins with the ON Clause

Use the ON clause to specify a join condition. With this, you can 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  
ON      (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	144 Vargas	50	50	1500
5	143 Matos	50	50	1500
6	142 Davies	50	50	1500
7	141 Rajs	50	50	1500
8	124 Mourgos	50	50	1500
9	103 Hunold	60	60	1400
10	104 Ernst	60	60	1400
11	107 Lorentz	60	60	1400

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Retrieving Records 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. The table alias is necessary to qualify the matching column_names.

You can also use the ON clause to join columns that have different names. The parenthesis around the joined columns, as in the example in the slide, (e.department_id = d.department_id) is optional. So, even ON e.department_id = d.department_id will work.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a '_1' to differentiate between the two department_ids.

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
9	142	South San Francisco	Shipping

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Three-Way Joins with the ON Clause

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.

Note: The code example in the slide can also be accomplished with the USING clause:

```
SELECT e.employee_id, l.city, d.department_name
FROM   employees e
JOIN   departments d
      USING (department_id)
JOIN   locations l
      USING (location_id)
```

Applying Additional Conditions to a Join

Use the `AND` clause or the `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  
ON     (e.department_id = d.department_id)  
AND    e.manager_id = 149 ;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id)  
WHERE  e.manager_id = 149 ;
```

ORACLE

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.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- **Self-join**
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Joining a Table to Itself

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...

**MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER table.**

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 the 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 worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON     (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Self-Joins Using the ON Clause

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.

Note: The parenthesis around the joined columns as in the example in the slide, (e.manager_id = m.employee_id) is **optional**. So, even ON e.manager_id = m.employee_id will work.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- **Nonequijoins**
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Nonequijoins

EMPLOYEES

R	LAST_NAME	R	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
...			
19	Taylor		8600
20	Grant		7000

JOB_GRADES

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

The **JOB_GRADES** table defines the **LOWEST_SAL** and **HIGHEST_SAL** range of values for each **GRADE_LEVEL**. Therefore, the **GRADE_LEVEL** column can be used to assign grades to each employee.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Nonequijoins

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. The **SALARY** column in the **EMPLOYEES** table ranges between the values in the **LOWEST_SAL** and **HIGHEST_SAL** columns of the **JOB_GRADES** table. Therefore, each employee can be graded based on their salary. The relationship is obtained using an operator other than the equality (=) operator.

Retrieving Records with Nonequijoins

```
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;
```

R	LAST_NAME	R	SALARY	R	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	
9	Fay		6000	C	
10	Grant		7000	C	

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Retrieving Records with Nonequijoins

The example in the slide 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_GRADES 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 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 \leq and \geq) can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using the BETWEEN condition. The Oracle server translates the BETWEEN condition to a pair of AND conditions. Therefore, using BETWEEN has no performance benefits, but should be used only for logical simplicity.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Returning Records with No Direct Match Using 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

There are no employees in department 190.

Employee "Grant" has not been assigned a department ID.

Equijoin with 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

...

18	80	Abel
19	80	Taylor

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Returning Records with No Direct Match Using OUTER Joins

If a row does not satisfy a join condition, the row does not appear in the query result.

In the slide example, a simple equijoin condition is used on the EMPLOYEES and DEPARTMENTS tables to return the result on the right. The result set does not contain the following:

- Department ID 190, because there are no employees with that department ID recorded in the EMPLOYEES table
- The employee with the last name of Grant, because this employee has not been assigned a department ID

To return the department record that does not have any employees, or employees that do not have an assigned department, 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) table 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	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping
...			
16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

LEFT OUTER JOIN

This query retrieves all the rows in the EMPLOYEES table, which is the left table, even if there is no match in the DEPARTMENTS table.

RIGHT OUTER JOIN

```
SELECT e.last_name, d.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
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

ORACLE

Copyright © 2009, Oracle. All rights reserved.

RIGHT OUTER JOIN

This query retrieves all the rows in the DEPARTMENTS table, which is the table at 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
4	Higgins	110	Accounting

...

17	Zlotkey	80	Sales
18	Abel	80	Sales
19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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.

Lesson Agenda

- Types of JOINS and its syntax
- Natural join:
 - USING clause
 - ON clause
- Self-join
- Nonequijoin
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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
- Always include a valid join condition if you want to avoid a Cartesian product.

ORACLE

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, therefore, 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
3	202	Fay	20
4	205	Higgins	110
...			
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:
20 x 8 = 160 rows

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700
...			
21	200	10	1800
22	201	20	1800
...			
159	176	80	1700
160	178	(null)	1700

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Generating a Cartesian Product

A Cartesian product is generated if a join condition is omitted. The example in the slide displays the employee last name and the department name from the EMPLOYEES and DEPARTMENTS tables. Because no join condition was 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
5	Fay	Administration

...

158	Vargas	Contracting
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.

The CROSS JOIN technique can be applied to many situations usefully. For example, to return total labor cost by office by month, even if month X has no labor cost, you can do a cross join of Offices with a table of all Months.

It is a good practice to explicitly state CROSS JOIN in your SELECT when you intend to create a Cartesian product. Therefore, it is very clear that you intend for this to happen and it is not the result of missing joins.

Quiz

The SQL:1999 standard join syntax supports the following types of joins. Which of these join types does Oracle join syntax support?

1. Equijoins
2. Nonequijoins
3. Left OUTER join
4. Right OUTER join
5. Full OUTER join
6. Self joins
7. Natural joins
8. Cartesian products

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answers: 1, 2, 3, 4, 6, 8

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

ORACLE

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 the display of all combinations of rows. This is done by either omitting the WHERE clause or by 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.
- Table aliases are sometimes mandatory to avoid column ambiguity.

Practice 6: Overview

This practice covers the following topics:

- Joining tables using an equijoin
- Performing outer and self-joins
- Adding conditions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 6: Overview

This practice is intended to give you experience in extracting data from more than one table using the SQL:1999-compliant joins.



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 the subqueries can solve
- List the types of subqueries
- Write single-row and multiple-row subqueries

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this lesson, you learn about the 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 also covers single-row subqueries and multiple-row subqueries.

Lesson Agenda

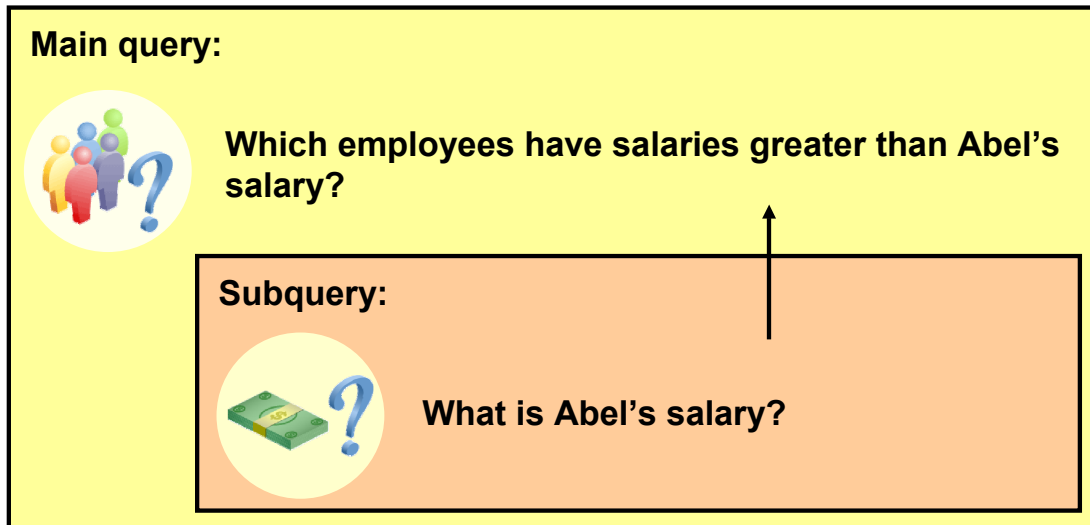
- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - HAVING clause with subqueries
- Multiple-row subqueries
 - Use ALL or ANY operator.
- Using the EXISTS operator
- Null values in a subquery

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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 *before* the main query (outer query).
- The result of the subquery is used by the main query.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Subquery Syntax

A subquery is a `SELECT` statement that is embedded in the 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 `IN`

Note: Comparison conditions fall into two classes: single-row operators (`>`, `=`, `>=`, `<`, `<>`, `<=`) and multiple-row operators (`IN`, `ANY`, `ALL`, `EXISTS`).

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
WHERE  salary > 11000
      (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 employee Abel.

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability. (However, the subquery can appear on either side of the comparison operator.)
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

ORACLE

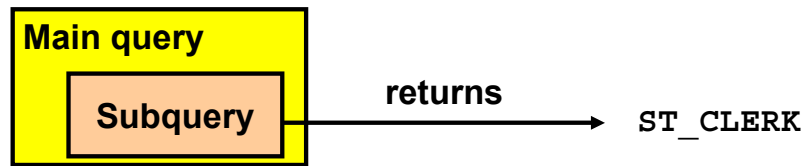
Copyright © 2009, Oracle. All rights reserved.

Guidelines for Using Subqueries

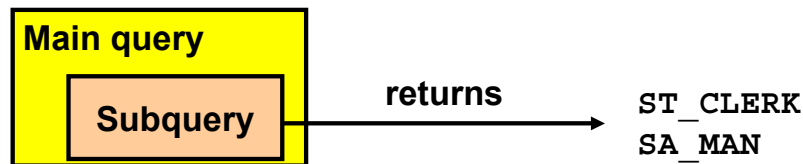
- A subquery must be enclosed in parentheses.
- Place the subquery on the right side of the comparison condition for readability. However, the subquery can appear on either side of the comparison operator.
- Two classes of comparison conditions are used in subqueries: single-row operators and multiple-row 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 11g: SQL Fundamentals II* course.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - HAVING clause with subqueries
- Multiple-row subqueries
 - Use ALL or ANY operator
- Using the EXISTS operator
- Null values in a subquery

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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
1	Rajs	ST_CLERK
2	Davies	ST_CLERK
3	Matos	ST_CLERK
4	Vargas	ST_CLERK

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = (SELECT job_id
                FROM employees
                WHERE last_name = 'Taylor')
AND salary > (SELECT salary
              FROM employees
              WHERE last_name = 'Taylor');
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

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 who do the same job as “Taylor,” but earn more salary than him.

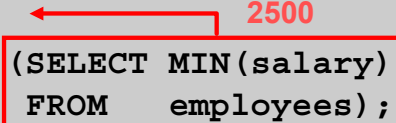
The example consists of three query blocks: the outer query and two inner queries. The inner query blocks are executed first, producing the query results SA_REP and 8600, respectively. The outer query block is then processed and uses the values that were returned by the inner queries to complete its search conditions.

Both inner queries return single values (SA_REP and 8600, 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
WHERE  salary = (SELECT MIN(salary)
                 FROM   employees);
```

A red arrow points from the value 2500 to the equals sign in the WHERE clause, indicating that the subquery result is used to compare the salary column.

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

ORACLE

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.

HAVING Clause with Subqueries

- The Oracle server executes the 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) > (SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

2500

	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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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.

Example:

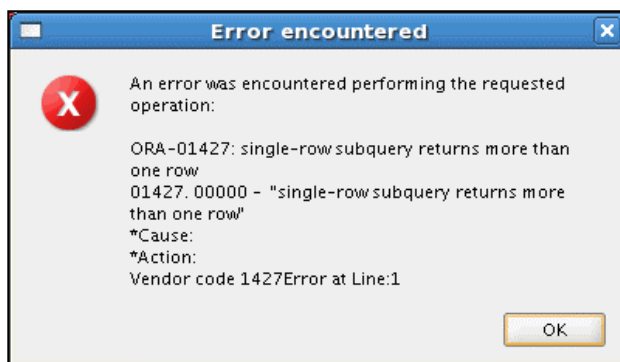
Find the job with the lowest average salary.

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
FROM employees
GROUP BY job_id);
```

	JOB_ID	AVG(SALARY)
1	ST_CLERK	2925

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.

What Is Wrong with This Statement?

A 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 results 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.

No Rows Returned by the Inner Query

```
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 rows because there is no employee named "Haas."

ORACLE

Copyright © 2009, Oracle. All rights reserved.

No Rows Returned by the Inner Query

Another 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 because there is no employee named Haas. Therefore, 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.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - `HAVING` clause with subqueries
- Multiple-row subqueries
 - Use `IN`, `ALL`, or `ANY`
- Using the `EXISTS` operator
- Null values in a subquery

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Compares a value to each value in a list or returned by a query. Evaluates to FALSE if the query returns no rows.
ALL	Must be preceded by =, !=, >, <, <=, >=. Compares a value to every value in a list or returned by a query. Evaluates to TRUE if the query returns no rows.

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

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400

...

9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the ANY Operator in Multiple-Row Subqueries

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

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the ALL Operator in Multiple-Row Subqueries

The ALL operator compares a value to *every* value returned by a subquery. The example in the slide 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.

Using the EXISTS Operator

```
SELECT * FROM departments
WHERE NOT EXISTS
(SELECT * FROM employees
WHERE employees.department_id=departments.department_id);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	190	Contracting	(null)	1700

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the EXISTS Operator

The EXISTS operator is used in queries where the query result depends on whether or not certain rows exist in a table. It evaluates to TRUE if the subquery returns at least one row.

The example in the slide displays departments that have no employees. For each row in the DEPARTMENTS table, the condition is checked whether there exists a row in the EMPLOYEES table that has the same department ID. In case no such row exists, the condition is satisfied for the row under consideration and it is selected. If there exists a corresponding row in the EMPLOYEES table, the row is not selected.

Lesson Agenda

- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
 - Group functions in a subquery
 - HAVING clause with subqueries
- Multiple-row subqueries
 - Use ALL or ANY operator
- Using the EXISTS operator
- Null values in a subquery

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Null Values in 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);
```


Null Values in 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);
```

Quiz

Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search values in the second query.

1. True
2. False

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answer: 1

Summary

In this lesson, you should have learned how to:

- Identify when a subquery can help solve a problem
- Write subqueries when a query is based on unknown values

```
SELECT    select_list
FROM      table
WHERE     expr operator
          (SELECT select_list
           FROM table);
```

ORACLE

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 the 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 7: Overview

This practice covers the following topics:

- Creating subqueries to query values based on unknown criteria
- Using subqueries to find out the values that exist in one set of data and not in another

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 7: Overview

In this practice, you write complex queries using nested `SELECT` statements.

For practice questions, you may want to create the inner query first. Make sure that it runs and produces the data that you anticipate before you code the outer query.

8

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this lesson, you learn how to write queries by using set operators.

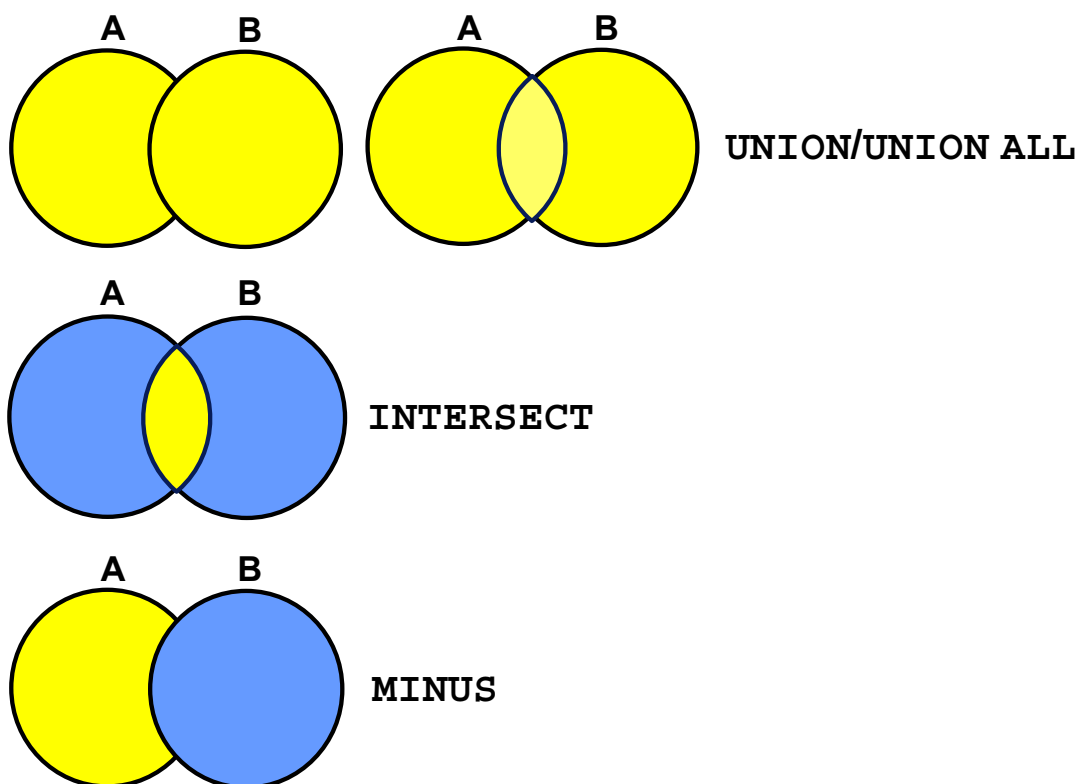
Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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	Rows from both queries after eliminating duplications
UNION ALL	Rows from both queries, including all duplications
INTERSECT	Rows that are common to both queries
MINUS	Rows in the first query that are not present in the second query

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.

Set Operator Guidelines

- The expressions in the `SELECT` lists must match in number.
- The data type of each column in the second query must match the data type of its corresponding column in the first query.
- Parentheses can be used to alter the sequence of execution.
- `ORDER BY` clause can appear only at the very end of the statement.

ORACLE

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 data type of columns in their `SELECT` list. The data type of the columns in the `SELECT` list of the queries in the compound query may not be exactly the same. The column in the second query must be in the same data type group (such as numeric or character) as the corresponding column in the first query.
- Set operators can be used in subqueries.
- You should use parentheses to specify the order of evaluation in queries that use the `INTERSECT` operator with other set operators. This ensures compliance with emerging SQL standards that will give the `INTERSECT` operator greater precedence than the other set operators.

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.

ORACLE

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 CHAR data type, of equal length, the returned values have the CHAR data type of that length. If the queries select values of CHAR with different lengths, the returned value is VARCHAR2 with the length of the larger CHAR value.
- If either or both of the queries select values of VARCHAR2 data type, the returned values have the VARCHAR2 data type.

If component queries select numeric data, the data type of the return values is determined by numeric precedence. If all queries select values of the NUMBER type, the returned values have the NUMBER data type. In queries using set operators, the Oracle server does not perform implicit conversion across data type groups. Therefore, if the corresponding expressions of component queries resolve to both character data and numeric data, the Oracle server returns an error.

Lesson Agenda

- Set Operators: Types and guidelines
- **Tables used in this lesson**
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Tables Used in This Lesson

Two tables are used in this lesson: the **EMPLOYEES** table and the **JOB_HISTORY** table.

You are already familiar with the **EMPLOYEES** table that stores employee details such as a unique identification number, email address, job identification (such as **ST_CLERK**, **SA_REP**, and so on), salary, manager, and so on.

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_id** (such as **ST_CLERK**, **SA_REP**, and so on), 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.

```
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		

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
```

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

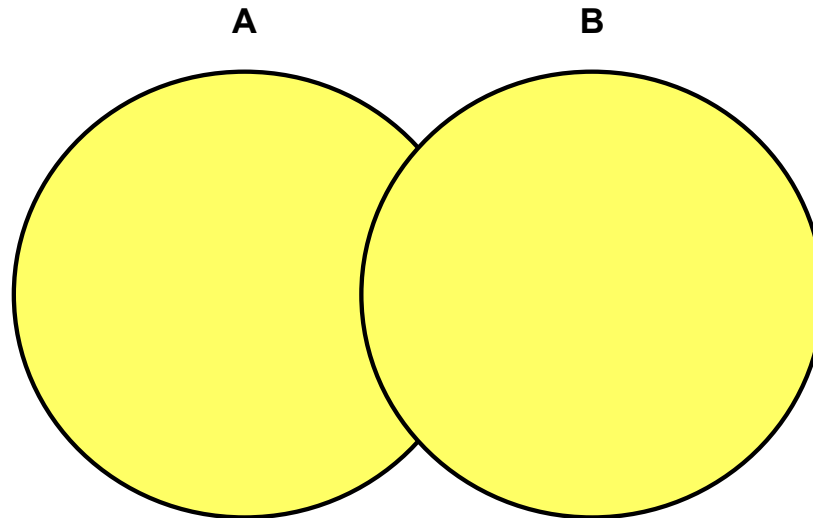
Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- **UNION and UNION ALL operator**
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

UNION Operator



The **UNION** operator returns rows 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 being selected must be the same.
- The data types of the columns being selected must be in the same data type group (such as numeric or character).
- 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.
- By default, the output is sorted in ascending order of the columns of the **SELECT** clause.

Using the UNION Operator

Display the current and previous job details of all employees.
Display each employee 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
...		
22	200	AC_ACCOUNT
23	200	AD_ASST
...		
27	205	AC_MGR
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:

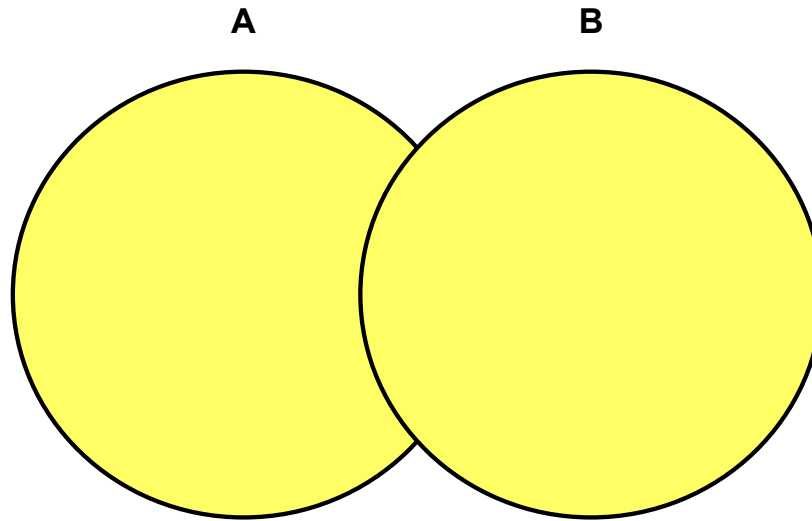
```
SELECT employee_id, job_id, department_id
FROM   employees
UNION
SELECT employee_id, job_id, department_id
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
...			
22	200	AC_ACCOUNT	90
23	200	AD_ASST	10
24	200	AD_ASST	90
...			
29	206	AC_ACCOUNT	110

Using the UNION Operator (continued)

In the preceding output, employee 200 appears three times. Why? Note 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 rows 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.

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
...		
17	149 SA_MAN	80
18	174 SA_REP	80
19	176 SA_REP	80
20	176 SA_MAN	80
21	176 SA_REP	80
22	178 SA_REP	(null)
23	200 AD_ASST	10
...		
30	206 AC_ACCOUNT	110

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the UNION ALL Operator

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 in 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):

176 SA_REP	80
------------	----

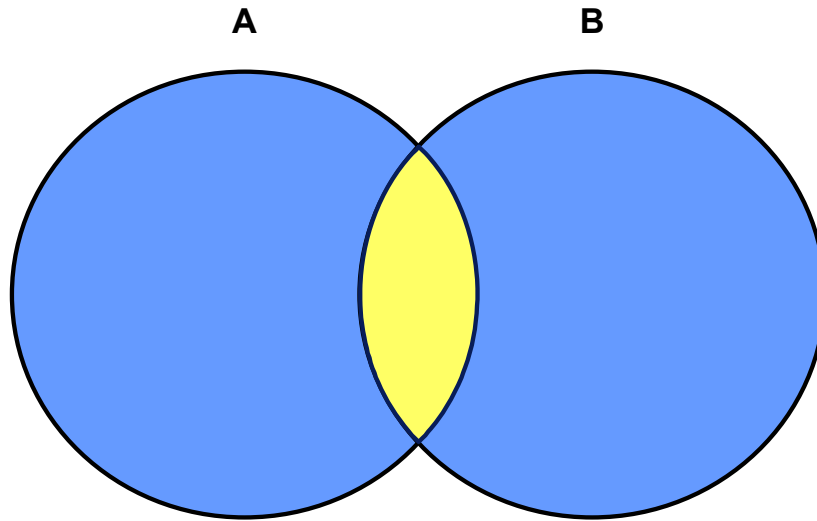
Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- **INTERSECT operator**
- MINUS operator
- Matching the SELECT statements
- Using ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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, however, 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 their previous one (that is, they changed jobs but have now gone back to doing the same job they did previously).

```
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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the INTERSECT Operator

In the example in this slide, the query returns only those 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	DEPARTMENT_ID
1	176	SA_REP	80

Employee 200 is no longer part of the results because the EMPLOYEES.DEPARTMENT_ID value is different from the JOB_HISTORY.DEPARTMENT_ID value.

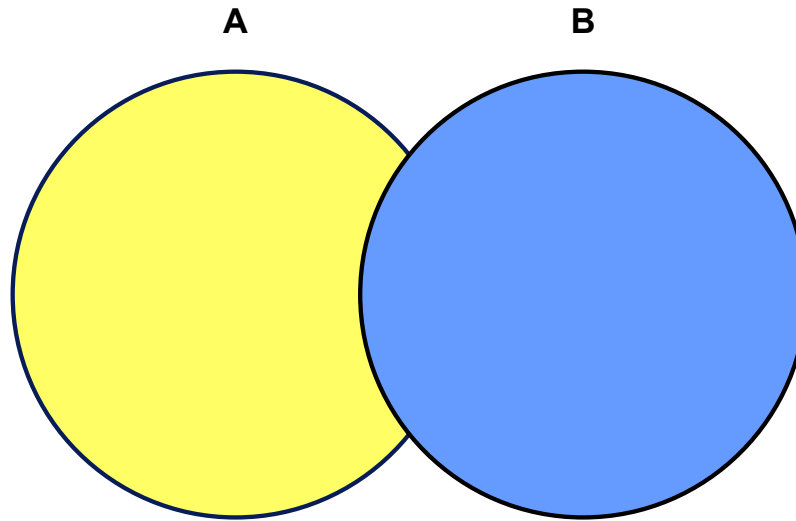
Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- **MINUS operator**
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

MINUS Operator



The **MINUS** operator returns all the distinct rows selected by the first query, but not present in the second query result set.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

MINUS Operator

Use the **MINUS** operator to return all distinct rows selected by the first query, but not present in the second query result set (the first **SELECT** statement **MINUS** the second **SELECT** statement).

Note: The number of columns must be the same and the data types of the columns being selected by the **SELECT** statements in the queries must belong to the same data type group in all the **SELECT** statements used in the query. The names of the columns, however, need not be identical.

Using the 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
1	100
2	103
3	104

...

13	202
14	205
15	206

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the MINUS Operator

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.

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- **Matching the SELECT statements**
- Using ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Matching the SELECT Statements

- Using the UNION operator, display the location ID, department name, and the state where it is located.
- You must match the data type (using the TO_CHAR function or any other conversion functions) when columns do not exist in one or the other table.

```
SELECT location_id, department_name "Department",  
       TO_CHAR(NULL) "Warehouse location"  
FROM departments  
UNION  
SELECT location_id, TO_CHAR(NULL) "Department",  
       state_province  
FROM locations;
```

ORACLE

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 the dummy columns and the data type conversion functions to comply with this rule. In the slide, the name, Warehouse location, is given as the dummy column heading. The TO_CHAR function is used in the first query to match the VARCHAR2 data type of the state_province column that is retrieved by the second query. Similarly, the TO_CHAR function in the second query is used to match the VARCHAR2 data type of the department_name column that is retrieved by the first query.

The output of the query is shown:

	LOCATION_ID	Department	Warehouse location
1	1400	IT	(null)
2	1400	(null)	Texas
3	1500	Shipping	(null)
4	1500	(null)	California
5	1700	Accounting	(null)
6	1700	Administration	(null)
7	1700	Contracting	(null)
8	1700	Executive	(null)

...

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
4	101	AD_VP	17000
5	102	AD_VP	17000
...			
29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

ORACLE

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 results shown in the slide, each row in the output that corresponds to a record from the JOB_HISTORY table contains a 0 in the SALARY column.

Lesson Agenda

- Set Operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching the SELECT statements
- Using the ORDER BY clause in set operations

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the ORDER BY Clause in Set Operations

- The ORDER BY clause can appear only once at the end of the compound query.
- Component queries cannot have individual ORDER BY clauses.
- The ORDER BY clause recognizes only the columns of the first SELECT query.
- By default, the first column of the first SELECT query is used to sort the output in an ascending order.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the ORDER BY Clause in Set Operations

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. By default, the output is sorted in ascending order in the first column of the first SELECT query.

Note: The ORDER BY clause does not recognize the column names of the second SELECT query. To avoid confusion over column names, it is a common practice to ORDER BY column positions.

For example, in the following statement, the output will be shown in ascending order of job_id.

```
SELECT employee_id, job_id, salary
FROM   employees
UNION
SELECT employee_id, job_id, 0
FROM   job_history
ORDER BY 2;
```

If you omit ORDER BY, by default, the output will be sorted in ascending order of employee_id. You cannot use the columns from the second query to sort the output.

Quiz

Identify the set operator guidelines.

1. The expressions in the `SELECT` lists must match in number.
2. Parentheses may not be used to alter the sequence of execution.
3. The data type of each column in the second query must match the data type of its corresponding column in the first query.
4. The `ORDER BY` clause can be used only once in a compound query, unless a `UNION ALL` operator is used.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answers: 1, 3

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Summary

- The `UNION` operator returns all the distinct rows selected by each query in the compound 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 8: Overview

In this practice, you create reports by using:

- The UNION operator
- The INTERSECT operator
- The MINUS operator

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 8: Overview

In this practice, you write queries using the set operators.

