



Integrated Cloud Applications & Platform Services

Unauthorized reproduction or distribution prohibited. Copyright © 2014, Oracle and/or its affiliates.



Oracle Database 12c: Introduction to SQL - Cloud Edition (WDP only)

Student Guide – Volume I

D99738GC20

Edition 2.0 | March 2017 | D99775

Learn more from Oracle University at education.oracle.com

ORACLE®

Author

Apoorva Srinivas

**Technical Contributors
and Reviewers**

Nancy Greenberg
Suresh Rajan
Peckkwai Yap
Bryan Roberts
Sharath Bhujani

Editors

Aju Kumar
Chandrika Kennedy
Kavita Saini

Graphic Designers

Prakash Dharmalingam
Kavya Bellur

Publishers

Asief Baig
Giri Venugopal
Jayanthy Keshavamurthy
Raghunath M
Srividya Rameshkumar
Sujatha Nagendra
Michael Sebastian

Copyright © 2017, Oracle and/or its affiliates. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Introduction

- Lesson Objectives 1-2
- Lesson Agenda 1-3
- Course Objectives 1-4
- Lesson Agenda 1-5
- Oracle Database 12c: Focus Areas 1-6
- Oracle Database 12c 1-7
- Lesson Agenda 1-9
- Relational and Object Relational Database Management Systems 1-10
- Data Storage on Different Media 1-12
- Relational Database Concept 1-13
- Definition of a Relational Database 1-14
- Data Models 1-15
- Entity Relationship Model 1-16
- Entity Relationship Modeling Conventions 1-18
- Relating Multiple Tables 1-20
- Relational Database Terminology 1-22
- Lesson Agenda 1-24
- Tables Used in This Course 1-26
- Tables Used in the Course 1-28
- Lesson Agenda 1-29
- Using SQL to Query Your Database 1-30
- How SQL Works 1-31
- SQL Statements Used in the Course 1-32
- Development Environments for SQL 1-33
- Introduction to Oracle Live SQL 1-34
- Lesson Agenda 1-35
- What is Oracle SQL Developer? 1-36
- Specifications of SQL Developer 1-37
- SQL Developer 3.2 Interface 1-38
- Creating a Database Connection 1-40
- Browsing Database Objects 1-43
- Displaying the Table Structure 1-44
- Browsing Files 1-45
- Creating a Schema Object 1-46

Creating a New Table: Example	1-47
Using SQL Worksheet	1-48
Executing SQL Statements	1-52
Saving SQL Scripts	1-53
Executing Saved Script Files: Method 1	1-54
Executing Saved Script Files: Method 2	1-55
Formatting the SQL Code	1-56
Using Snippets	1-57
Using Snippets: Example	1-58
Using the Recycle Bin	1-59
Debugging Procedures and Functions	1-60
Database Reporting	1-61
Creating a User-Defined Report	1-62
Search Engines and External Tools	1-63
Setting Preferences	1-64
Resetting the SQL Developer Layout	1-66
Data Modeler in SQL Developer	1-67
Lesson Agenda	1-68
Oracle Database Documentation	1-69
Additional Resources	1-70
Summary	1-71
Practice 1: Overview	1-72

2 Retrieving Data Using the SQL SELECT Statement

Objectives	2-2
Lesson Agenda	2-3
Basic SELECT Statement	2-5
Selecting All Columns	2-6
Selecting Specific Columns	2-7
Selecting from DUAL	2-8
Writing SQL Statements	2-9
Column Heading Defaults	2-10
Lesson Agenda	2-11
Arithmetic Expressions	2-12
Using Arithmetic Operators	2-13
Operator Precedence	2-14
Defining a Null Value	2-15
Null Values in Arithmetic Expressions	2-16
Lesson Agenda	2-17
Defining a Column Alias	2-18
Using Column Aliases	2-19

Lesson Agenda	2-20
Concatenation Operator	2-21
Literal Character Strings	2-22
Using Literal Character Strings	2-23
Alternative Quote (q) Operator	2-24
Duplicate Rows	2-25
Lesson Agenda	2-26
Displaying Table Structure	2-27
Using the DESCRIBE Command	2-28
Quiz	2-29
Summary	2-30
Practice 2: Overview	2-31

3 Restricting and Sorting Data

Objectives	3-2
Lesson Agenda	3-3
Limiting Rows by Using a Selection	3-4
Limiting Rows That Are Selected	3-5
Using the WHERE Clause	3-6
Character Strings and Dates	3-7
Comparison Operators	3-8
Using Comparison Operators	3-9
Range Conditions Using the BETWEEN Operator	3-10
Using the IN Operator	3-11
Pattern Matching Using the LIKE Operator	3-12
Combining Wildcard Symbols	3-13
Using NULL Conditions	3-14
Defining Conditions Using Logical Operators	3-15
Using the AND Operator	3-16
Using the OR Operator	3-17
Using the NOT Operator	3-18
Lesson Agenda	3-19
Rules of Precedence	3-20
Lesson Agenda	3-22
Using the ORDER BY Clause	3-23
Sorting	3-24
Lesson Agenda	3-26
SQL Row Limiting Clause	3-27
Using SQL Row Limiting Clause in a Query	3-28
SQL Row Limiting Clause: Example	3-29
Lesson Agenda	3-30

Substitution Variables	3-31
Using the Single-Ampersand Substitution Variable	3-33
Character and Date Values with Substitution Variables	3-35
Specifying Column Names, Expressions, and Text	3-36
Using the Double-Ampersand Substitution Variable	3-37
Using the Ampersand Substitution Variable in SQL*Plus	3-38
Lesson Agenda	3-39
Using the DEFINE Command	3-40
Using the VERIFY Command	3-41
Quiz	3-42
Summary	3-43
Practice 3: Overview	3-44

4 Using Single-Row Functions to Customize Output

Objectives	4-2
HR Application Scenario	4-3
Lesson Agenda	4-4
SQL Functions	4-5
Two Types of SQL Functions	4-6
Single-Row Functions	4-7
Lesson Agenda	4-9
Character Functions	4-10
Case-Conversion Functions	4-12
Using Case-Conversion Functions	4-13
Character-Manipulation Functions	4-14
Using Character-Manipulation Functions	4-15
Lesson Agenda	4-16
Nesting Functions	4-17
Nesting Functions: Example	4-18
Lesson Agenda	4-19
Numeric Functions	4-20
Using the ROUND Function	4-21
Using the TRUNC Function	4-22
Using the MOD Function	4-23
Lesson Agenda	4-24
Working with Dates	4-25
RR Date Format	4-26
Using the SYSDATE Function	4-28
Using the CURRENT_DATE and CURRENT_TIMESTAMP Functions	4-29
Arithmetic with Dates	4-30
Using Arithmetic Operators with Dates	4-31

Lesson Agenda 4-32
Date-Manipulation Functions 4-33
Using Date Functions 4-34
Using ROUND and TRUNC Functions with Dates 4-35
Quiz 4-36
Summary 4-37
Practice 4: Overview 4-38

5 Using Conversion Functions and Conditional Expressions

Objectives 5-2
Lesson Agenda 5-3
Conversion Functions 5-4
Implicit Data Type Conversion of Strings 5-5
Implicit Data Type Conversion to Strings 5-6
Explicit Data Type Conversion 5-7
Lesson Agenda 5-9
Using the TO_CHAR Function with Dates 5-10
Elements of the Date Format Model 5-11
Using the TO_CHAR Function with Dates 5-15
Using the TO_CHAR Function with Numbers 5-16
Using the TO_NUMBER and TO_DATE Functions 5-19
Using TO_CHAR and TO_DATE Functions with the RR Date Format 5-21
Lesson Agenda 5-22
General Functions 5-23
NVL Function 5-24
Using the NVL Function 5-25
Using the NVL2 Function 5-26
Using the NULLIF Function 5-27
Using the COALESCE Function 5-28
Lesson Agenda 5-30
Conditional Expressions 5-31
CASE Expression 5-32
Using the CASE Expression 5-33
Searched CASE Expression 5-34
DECODE Function 5-35
Using the DECODE Function 5-36
Quiz 5-38
Summary 5-39
Practice 5: Overview 5-40

6 Reporting Aggregated Data Using the Group Functions

- Objectives 6-2
- Lesson Agenda 6-3
- Group Functions 6-4
- Types of Group Functions 6-5
- Group Functions: Syntax 6-6
- Using the AVG and SUM Functions 6-7
- Using the MIN and MAX Functions 6-8
- Using the COUNT Function 6-9
- Using the DISTINCT Keyword 6-10
- Group Functions and Null Values 6-11
- Lesson Agenda 6-12
- Creating Groups of Data 6-13
- Creating Groups of Data: GROUP BY Clause Syntax 6-14
- Using the GROUP BY Clause 6-15
- Grouping by More Than One Column 6-17
- Using the GROUP BY Clause on Multiple Columns 6-18
- Illegal Queries Using Group Functions 6-19
- Restricting Group Results 6-21
- Restricting Group Results with the HAVING Clause 6-22
- Using the HAVING Clause 6-23
- Lesson Agenda 6-25
- Nesting Group Functions 6-26
- Quiz 6-27
- Summary 6-28
- Practice 6: Overview 6-29

7 Displaying Data from Multiple Tables Using Joins

- Objectives 7-2
- Lesson Agenda 7-3
- Why Join? 7-4
- Obtaining Data from Multiple Tables 7-5
- Types of Joins 7-6
- Joining Tables Using SQL:1999 Syntax 7-7
- Lesson Agenda 7-8
- Creating Natural Joins 7-9
- Retrieving Records with Natural Joins 7-10
- Creating Joins with the USING Clause 7-11
- Joining Column Names 7-12
- Retrieving Records with the USING Clause 7-13
- Qualifying Ambiguous Column Names 7-14

Using Table Aliases with the USING Clause	7-15
Creating Joins with the ON Clause	7-16
Retrieving Records with the ON Clause	7-17
Creating Three-Way Joins	7-18
Applying Additional Conditions to a Join	7-19
Lesson Agenda	7-20
Joining a Table to Itself	7-21
Self-Joins Using the ON Clause	7-22
Lesson Agenda	7-23
Nonequijoins	7-24
Retrieving Records with Nonequijoins	7-25
Lesson Agenda	7-26
Returning Records with No Direct Match Using OUTER Joins	7-27
INNER Versus OUTER Joins	7-28
LEFT OUTER JOIN	7-29
RIGHT OUTER JOIN	7-30
FULL OUTER JOIN	7-31
Lesson Agenda	7-32
Cartesian Products	7-33
Generating a Cartesian Product	7-34
Creating Cross Joins	7-35
Quiz	7-36
Summary	7-37
Practice 7: Overview	7-38

8 Using Subqueries to Solve Queries

Objectives	8-2
Lesson Agenda	8-3
Using a Subquery to Solve a Problem	8-4
Subquery Syntax	8-5
Using a Subquery	8-6
Rules and Guidelines for Using Subqueries	8-7
Types of Subqueries	8-8
Lesson Agenda	8-9
Single-Row Subqueries	8-10
Executing Single-Row Subqueries	8-11
Using Group Functions in a Subquery	8-12
HAVING Clause with Subqueries	8-13
What Is Wrong with This Statement?	8-14
No Rows Returned by the Inner Query	8-15
Lesson Agenda	8-16

Multiple-Row Subqueries 8-17
Using the ANY Operator in Multiple-Row Subqueries 8-18
Using the ALL Operator in Multiple-Row Subqueries 8-19
Multiple-Column Subqueries 8-20
Multiple-Column Subquery: Example 8-21
Lesson Agenda 8-22
Null Values in a Subquery 8-23
Quiz 8-25
Summary 8-26
Practice 8: Overview 8-27

9 Using Set Operators

Objectives 9-2
Lesson Agenda 9-3
Set Operators 9-4
Set Operator Rules 9-5
Oracle Server and Set Operators 9-6
Lesson Agenda 9-7
Tables Used in This Lesson 9-8
Lesson Agenda 9-12
UNION Operator 9-13
Using the UNION Operator 9-14
UNION ALL Operator 9-15
Using the UNION ALL Operator 9-16
Lesson Agenda 9-17
INTERSECT Operator 9-18
Using the INTERSECT Operator 9-19
Lesson Agenda 9-20
MINUS Operator 9-21
Using the MINUS Operator 9-22
Lesson Agenda 9-23
Matching SELECT Statements 9-24
Matching the SELECT Statement: Example 9-25
Lesson Agenda 9-26
Using the ORDER BY Clause in Set Operations 9-27
Quiz 9-28
Summary 9-29
Practice 9: Overview 9-30

10 Managing Tables Using DML Statements

Objectives 10-2

HR Application Scenario	10-3
Lesson Agenda	10-4
Data Manipulation Language	10-5
Adding a New Row to a Table	10-6
INSERT Statement Syntax	10-7
Inserting New Rows	10-8
Inserting Rows with Null Values	10-9
Inserting Special Values	10-10
Inserting Specific Date and Time Values	10-11
Creating a Script	10-12
Copying Rows from Another Table	10-13
Lesson Agenda	10-14
Changing Data in a Table	10-15
UPDATE Statement Syntax	10-16
Updating Rows in a Table	10-17
Updating Two Columns with a Subquery	10-18
Updating Rows Based on Another Table	10-19
Lesson Agenda	10-20
Removing a Row from a Table	10-21
DELETE Statement	10-22
Deleting Rows from a Table	10-23
Deleting Rows Based on Another Table	10-24
TRUNCATE Statement	10-25
Lesson Agenda	10-26
Database Transactions	10-27
Database Transactions: Start and End	10-28
Advantages of COMMIT and ROLLBACK Statements	10-29
Explicit Transaction Control Statements	10-30
Rolling Back Changes to a Marker	10-31
Implicit Transaction Processing	10-32
State of Data Before COMMIT or ROLLBACK	10-34
State of Data After COMMIT	10-35
Committing Data	10-36
State of Data After ROLLBACK	10-37
State of Data After ROLLBACK: Example	10-38
Statement-Level Rollback	10-39
Lesson Agenda	10-40
Read Consistency	10-41
Implementing Read Consistency	10-42
Lesson Agenda	10-43
FOR UPDATE Clause in a SELECT Statement	10-44

FOR UPDATE Clause: Examples 10-45
LOCK TABLE Statement 10-47
Quiz 10-48
Summary 10-49
Practice 10: Overview 10-50

11 Introduction to Data Definition Language

Objectives 11-2
HR Application Scenario 11-3
Lesson Agenda 11-4
Database Objects 11-5
Naming Rules for Tables and Columns 11-6
Lesson Agenda 11-7
CREATE TABLE Statement 11-8
Creating Tables 11-9
Lesson Agenda 11-10
Data Types 11-11
Datetime Data Types 11-13
DEFAULT Option 11-14
Lesson Agenda 11-15
Including Constraints 11-16
Constraint Guidelines 11-17
Defining Constraints 11-18
Defining Constraints: Example 11-19
NOT NULL Constraint 11-20
UNIQUE Constraint 11-21
PRIMARY KEY Constraint 11-23
FOREIGN KEY Constraint 11-24
FOREIGN KEY Constraint: Keywords 11-26
CHECK Constraint 11-27
CREATE TABLE: Example 11-28
Violating Constraints 11-29
Lesson Agenda 11-31
Creating a Table Using a Subquery 11-32
Lesson Agenda 11-34
ALTER TABLE Statement 11-35
Adding a Column 11-37
Modifying a Column 11-38
Dropping a Column 11-39
SET UNUSED Option 11-40
Read-Only Tables 11-42

Lesson Agenda 11-43
Dropping a Table 11-44
Quiz 11-45
Summary 11-46
Practice 11: Overview 11-47

12 Introduction to Data Dictionary Views

Objectives 12-2
Lesson Agenda 12-3
Why Data Dictionary? 12-4
Data Dictionary 12-5
Data Dictionary Structure 12-6
How to Use Dictionary Views 12-8
USER_OBJECTS and ALL_OBJECTS Views 12-9
USER_OBJECTS View 12-10
Lesson Agenda 12-11
Table Information 12-12
Column Information 12-13
Constraint Information 12-15
USER_CONSTRAINTS: Example 12-16
Querying USER_CONS_COLUMNS 12-17
Lesson Agenda 12-18
Adding Comments to a Table 12-19
Quiz 12-20
Summary 12-21
Practice 12: Overview 12-22

13 Creating Sequences, Synonyms, and Indexes

Objectives 13-2
Lesson Agenda 13-3
E-Commerce Scenario 13-4
Database Objects 13-5
Referencing Another User's Tables 13-6
Sequences 13-7
CREATE SEQUENCE Statement: Syntax 13-8
Creating a Sequence 13-10
NEXTVAL and CURRVAL Pseudocolumns 13-11
Using a Sequence 13-13
SQL Column Defaulting Using a Sequence 13-14
Caching Sequence Values 13-15
Modifying a Sequence 13-16

Guidelines for Modifying a Sequence	13-17
Sequence Information	13-18
Lesson Agenda	13-19
Synonyms	13-20
Creating a Synonym for an Object	13-21
Creating and Removing Synonyms	13-22
Synonym Information	13-23
Lesson Agenda	13-24
Indexes	13-25
How Are Indexes Created?	13-26
Creating an Index	13-27
CREATE INDEX with the CREATE TABLE Statement	13-28
Function-Based Indexes	13-30
Creating Multiple Indexes on the Same Set of Columns	13-31
Creating Multiple Indexes on the Same Set of Columns: Example	13-32
Index Information	13-33
USER_INDEXES: Examples	13-34
Querying USER_IND_COLUMNS	13-35
Removing an Index	13-36
Quiz	13-37
Summary	13-38
Practice 13: Overview	13-39

14 Creating Views

Objectives	14-2
Lesson Agenda	14-3
Why Views?	14-4
Database Objects	14-5
What Is a View?	14-6
Advantages of Views	14-7
Simple Views and Complex Views	14-8
Lesson Agenda	14-9
Creating a View	14-10
Retrieving Data from a View	14-13
Modifying a View	14-14
Creating a Complex View	14-15
View Information	14-16
Lesson Agenda	14-17
Rules for Performing DML Operations on a View	14-18
Rules for Performing Modify Operations on a View	14-19
Rules for Performing Insert Operations Through a View	14-20

Using the WITH CHECK OPTION Clause	14-21
Denying DML Operations	14-22
Lesson Agenda	14-24
Removing a View	14-25
Quiz	14-26
Summary	14-27
Practice 14: Overview	14-28

15 Managing Schema Objects

Objectives	15-2
Lesson Agenda	15-3
Adding a Constraint Syntax	15-4
Adding a Constraint	15-5
Dropping a Constraint	15-6
Dropping a Constraint ONLINE	15-7
ON DELETE Clause	15-8
Cascading Constraints	15-9
Renaming Table Columns and Constraints	15-11
Disabling Constraints	15-12
Enabling Constraints	15-13
Constraint States	15-14
Deferring Constraints	15-15
Difference Between INITIALLY DEFERRED and INITIALLY IMMEDIATE	15-16
DROP TABLE ... PURGE	15-18
Lesson Agenda	15-19
Using Temporary Tables	15-20
Creating a Temporary Table	15-21
Lesson Agenda	15-22
External Tables	15-23
Creating a Directory for the External Table	15-24
Creating an External Table	15-26
Creating an External Table by Using ORACLE_LOADER	15-28
Querying External Tables	15-29
Creating an External Table by Using ORACLE_DATAPUMP: Example	15-30
Quiz	15-31
Summary	15-32
Practice 15: Overview	15-33

16 Retrieving Data by Using Subqueries

Objectives	16-2
Lesson Agenda	16-3

Retrieving Data by Using a Subquery as a Source 16-4
Lesson Agenda 16-6
Multiple-Column Subqueries 16-7
Column Comparisons 16-8
Pairwise Comparison Subquery 16-9
Nonpairwise Comparison Subquery 16-10
Lesson Agenda 16-11
Scalar Subquery Expressions 16-12
Scalar Subqueries: Examples 16-13
Lesson Agenda 16-14
Correlated Subqueries 16-15
Using Correlated Subqueries: Example 1 16-17
Using Correlated Subqueries: Example 2 16-18
Lesson Agenda 16-19
Using the EXISTS Operator 16-20
Find All Departments That Do Not Have Any Employees 16-22
Lesson Agenda 16-23
WITH Clause 16-24
WITH Clause: Example 16-25
Recursive WITH Clause 16-26
Recursive WITH Clause: Example 16-27
Quiz 16-28
Summary 16-29
Practice 16: Overview 16-30

17 Manipulating Data by Using Subqueries

Objectives 17-2
Lesson Agenda 17-3
Using Subqueries to Manipulate Data 17-4
Lesson Agenda 17-5
Inserting by Using a Subquery as a Target 17-6
Lesson Agenda 17-8
Using the WITH CHECK OPTION Keyword on DML Statements 17-9
Lesson Agenda 17-11
Correlated UPDATE 17-12
Using Correlated UPDATE 17-13
Correlated DELETE 17-15
Using Correlated DELETE 17-16
Summary 17-17
Practice 17: Overview 17-18

18 Controlling User Access

Objectives 18-2
Lesson Agenda 18-3
Controlling User Access 18-4
Privileges 18-5
System Privileges 18-6
Creating Users 18-7
User System Privileges 18-8
Granting System Privileges 18-9
Lesson Agenda 18-10
What is a Role? 18-11
Creating and Granting Privileges to a Role 18-12
Changing Your Password 18-13
Lesson Agenda 18-14
Object Privileges 18-15
Granting Object Privileges 18-17
Passing On Your Privileges 18-18
Confirming Granted Privileges 18-19
Lesson Agenda 18-20
Revoking Object Privileges 18-21
Quiz 18-23
Summary 18-24
Practice 18: Overview 18-25

19 Manipulating Data Using Advanced Queries

Objectives 19-2
Lesson Agenda 19-3
Explicit Default Feature: Overview 19-4
Using Explicit Default Values 19-5
Lesson Agenda 19-6
E-Commerce Scenario 19-7
Multitable INSERT Statements: Overview 19-8
Types of Multitable INSERT Statements 19-10
Multitable INSERT Statements 19-11
Unconditional INSERT ALL 19-13
Conditional INSERT ALL: Example 19-14
Conditional INSERT ALL 19-15
Conditional INSERT FIRST: Example 19-17
Conditional INSERT FIRST 19-18
Pivoting INSERT 19-19
Lesson Agenda 19-22

MERGE Statement 19-23
MERGE Statement Syntax 19-24
Merging Rows: Example 19-25
Lesson Agenda 19-28
FLASHBACK TABLE Statement 19-29
Using the FLASHBACK TABLE Statement 19-31
Lesson Agenda 19-32
Tracking Changes in Data 19-33
Flashback Query: Example 19-34
Flashback Version Query: Example 19-35
VERSIONS BETWEEN Clause 19-36
Quiz 19-37
Summary 19-39
Practice 19: Overview 19-40

20 Managing Data in Different Time Zones

Objectives 20-2
Lesson Agenda 20-3
E-Commerce Scenario 20-4
Time Zones 20-5
TIME_ZONE Session Parameter 20-6
CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP 20-7
Comparing Date and Time in a Session's Time Zone 20-8
DBTIMEZONE and SESSIONTIMEZONE 20-10
TIMESTAMP Data Types 20-11
TIMESTAMP Fields 20-12
Difference Between DATE and TIMESTAMP 20-13
Comparing TIMESTAMP Data Types 20-14
Lesson Agenda 20-15
INTERVAL Data Types 20-16
INTERVAL Fields 20-17
INTERVAL YEAR TO MONTH: Example 20-18
INTERVAL DAY TO SECOND Data Type: Example 20-20
Lesson Agenda 20-21
EXTRACT 20-22
TZ_OFFSET 20-23
FROM_TZ 20-25
TO_TIMESTAMP 20-26
TO_YMINTERVAL 20-27
TO_DSINTERVAL 20-28
Daylight Saving Time (DST) 20-29

Quiz 20-31

Summary 20-32

Practice 20: Overview 20-33

21 Oracle Cloud Overview

Agenda 21-2

What Is Cloud? 21-3

What Is Cloud Computing? 21-4

History: Cloud Evolution 21-5

Components of Cloud Computing 21-6

Characteristics of Cloud 21-7

Cloud Deployment Models 21-8

Cloud Service Models 21-9

Cloud Service Models: IaaS 21-10

Cloud Service Models: PaaS 21-11

Cloud Service Models: SaaS 21-12

Industry Shifting from On-Premises to Cloud 21-13

Oracle IaaS: Overview 21-15

Oracle PaaS: Overview 21-16

Oracle SaaS: Overview 21-17

Summary 21-18

A Table Descriptions

B Using SQL*Plus

Objectives B-2

SQL and SQL*Plus Interaction B-3

SQL Statements Versus SQL*Plus Commands B-4

SQL*Plus: Overview B-5

Logging in to SQL*Plus B-6

Displaying the Table Structure B-7

SQL*Plus Editing Commands B-9

Using LIST, n, and APPEND B-11

Using the CHANGE Command B-12

SQL*Plus File Commands B-13

Using the SAVE and START Commands B-14

SERVERROUTPUT Command B-15

Using the SQL*Plus SPOOL Command B-16

Using the AUTOTRACE Command B-17

Summary B-18

C Commonly Used SQL Commands

- Objectives C-2
Basic SELECT Statement C-3
SELECT Statement C-4
WHERE Clause C-5
ORDER BY Clause C-6
GROUP BY Clause C-7
Data Definition Language C-8
CREATE TABLE Statement C-9
ALTER TABLE Statement C-10
DROP TABLE Statement C-11
GRANT Statement C-12
Privilege Types C-13
REVOKE Statement C-14
TRUNCATE TABLE Statement C-15
Data Manipulation Language C-16
INSERT Statement C-17
UPDATE Statement Syntax C-18
DELETE Statement C-19
Transaction Control Statements C-20
COMMIT Statement C-21
ROLLBACK Statement C-22
SAVEPOINT Statement C-23
Joins C-24
Types of Joins C-25
Qualifying Ambiguous Column Names C-26
Natural Join C-27
Equijoins C-28
Retrieving Records with Equijoins C-29
Additional Search Conditions Using the AND and WHERE Operators C-30
Retrieving Records with Nonequijoins C-31
Retrieving Records by Using the USING Clause C-32
Retrieving Records by Using the ON Clause C-33
Left Outer Join C-34
Right Outer Join C-35
Full Outer Join C-36
Self-Join: Example C-37
Cross Join C-38
Summary C-39

D Generating Reports by Grouping Related Data

- Objectives D-2
- Review of Group Functions D-3
- Review of the GROUP BY Clause D-4
- Review of the HAVING Clause D-5
- GROUP BY with ROLLUP and CUBE Operators D-6
- ROLLUP Operator D-7
- ROLLUP Operator: Example D-8
- CUBE Operator D-9
- CUBE Operator: Example D-10
- GROUPING Function D-11
- GROUPING Function: Example D-12
- GROUPING SETS D-13
- GROUPING SETS: Example D-15
- Composite Columns D-17
- Composite Columns: Example D-19
- Concatenated Groupings D-21
- Concatenated Groupings: Example D-22
- Summary D-23

E Hierarchical Retrieval

- Objectives E-2
- Sample Data from the EMPLOYEES Table E-3
- Natural Tree Structure E-4
- Hierarchical Queries E-5
- Walking the Tree E-6
- Walking the Tree: From the Bottom Up E-8
- Walking the Tree: From the Top Down E-9
- Ranking Rows with the LEVEL Pseudocolumn E-10
- Formatting Hierarchical Reports Using LEVEL and LPAD E-11
- Pruning Branches E-13
- Summary E-14

F Writing Advanced Scripts

- Objectives F-2
- Using SQL to Generate SQL F-3
- Creating a Basic Script F-4
- Controlling the Environment F-5
- The Complete Picture F-6
- Dumping the Contents of a Table to a File F-7

Generating a Dynamic Predicate F-9
Summary F-11

G Oracle Database Architectural Components

Objectives G-2
Oracle Database Architecture: Overview G-3
Oracle Database Server Structures G-4
Connecting to the Database G-5
Interacting with an Oracle Database G-6
Oracle Memory Architecture G-8
Process Architecture G-10
Database Writer Process G-12
Log Writer Process G-13
Checkpoint Process G-14
System Monitor Process G-15
Process Monitor Process G-16
Oracle Database Storage Architecture G-17
Logical and Physical Database Structures G-19
Processing a SQL Statement G-21
Processing a Query G-22
Shared Pool G-23
Database Buffer Cache G-25
Program Global Area (PGA) G-26
Processing a DML Statement G-27
Redo Log Buffer G-29
Rollback Segment G-30
COMMIT Processing G-31
Summary of the Oracle Database Architecture G-33
Summary G-34

H Regular Expression Support

Objectives H-2
What Are Regular Expressions? H-3
Benefits of Using Regular Expressions H-4
Using the Regular Expressions Functions and Conditions in SQL and PL/SQL H-5
What are Metacharacters? H-6
Using Metacharacters with Regular Expressions H-7
Regular Expressions Functions and Conditions: Syntax H-9
Performing a Basic Search by Using the REGEXP_LIKE Condition H-10
Replacing Patterns by Using the REGEXP_REPLACE Function H-11
Finding Patterns by Using the REGEXP_INSTR Function H-12

Extracting Substrings by Using the REGEXP_SUBSTR Function	H-13
Subexpressions	H-14
Using Subexpressions with Regular Expression Support	H-15
Why Access the nth Subexpression?	H-16
REGEXP_SUBSTR: Example	H-17
Using the REGEXP_COUNT Function	H-18
Regular Expressions and Check Constraints: Examples	H-19
Quiz	H-20
Summary	H-21

Introduction

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cornelia Sirbu (vrabie.cornelia@gmail.com) has a non-transferable
license to use this Student Guide.

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 12c
- Discuss the theoretical and physical aspects of a relational database
- Describe Oracle server's implementation of relational database management system (RDBMS) and object relational database management system (ORDBMS)
- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Course objectives
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in the course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Course Objectives

After completing this course, you should be able to:

- Identify the major components of Oracle Database
- 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
- Run data definition language (DDL) statements to create and manage schema objects



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

This course offers you an introduction to the Oracle Database technology. Completing this course will equip you with essential SQL skills. Some of the tasks you can do with these skills include querying single and multiple tables, inserting data in tables, creating tables, and querying metadata.

Lesson Agenda

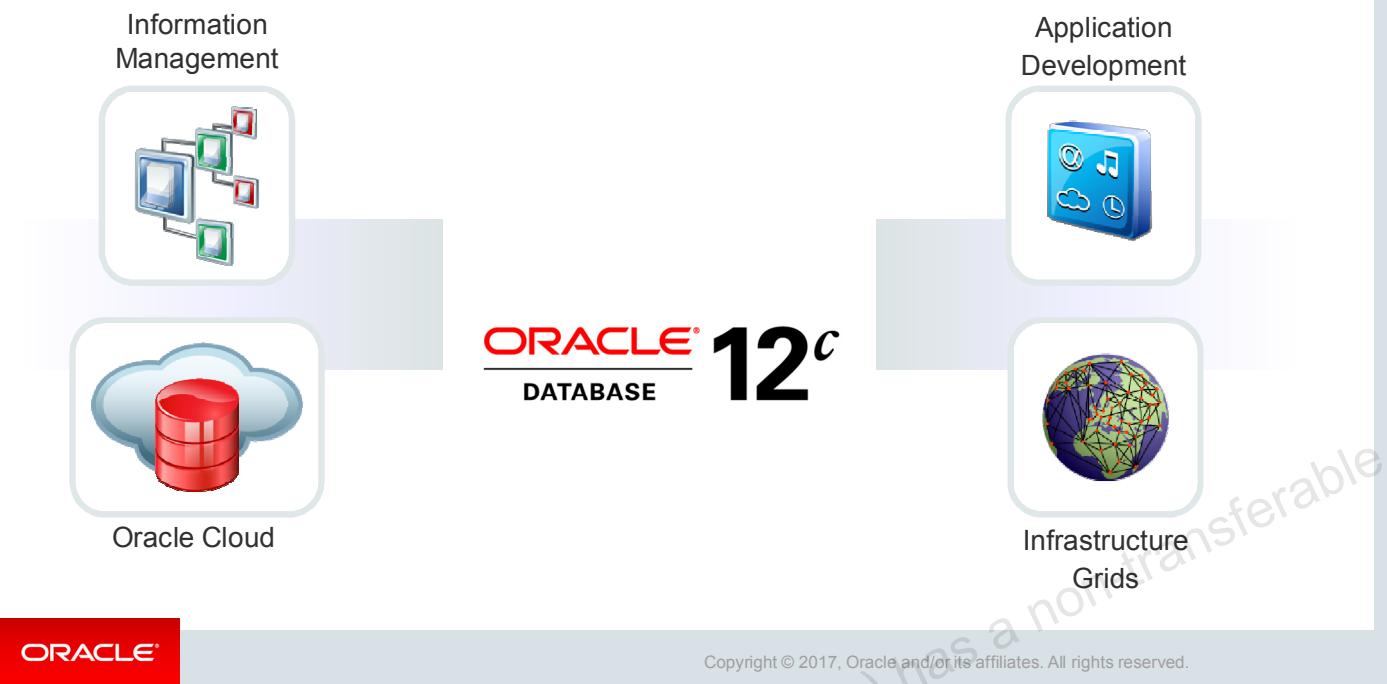
- Course objectives
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in the course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle Database 12c: Focus Areas



By using Oracle Database 12c, you can utilize the following features across focus areas:

- With the **Infrastructure Grid** technology of Oracle, you can pool 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 12c also helps you to consolidate and extend the benefits of grid computing and manage changes in a controlled and cost-effective manner.
- Oracle Database 12c enables **Information Management** by providing capabilities in content management, information integration, and information lifecycle management areas. You can manage content of advanced data types such as Extensible Markup Language (XML), text, spatial, multimedia, medical imaging, and semantic technologies using the features provided by Oracle.
- With Oracle Database 12c, you can manage all the major **Application Development** environments such as PL/SQL, Java/JDBC, .NET, Windows, PHP, SQL Developer, and Application Express.
- You can now plug into **Oracle Cloud** with Oracle Database 12c. This will help you to standardize, consolidate, and automate database services on the cloud.

Oracle Database 12c



Imagine you have an organization that needs 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 12c is designed 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 the Database Replay facility, the SQL Performance Analyzer, the Automatic SQL Tuning facility, and Real-Time Database Operations Monitoring.

Enterprise Manager Database Express 12c is a web-based tool for managing Oracle databases. It greatly simplifies database performance diagnostics by consolidating the relevant database performance screens into a view called Database Performance Hub. DBAs get a single, consolidated view of the current real-time and historical view of the database performance across multiple dimensions such as database load, monitored SQL and PL/SQL, and Active Session History (ASH) on a single page for the selected time period.

- **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.
- **Performance:** By using capabilities such as SecureFiles, Result Caches, and so on, you can greatly improve the performance of your database. Oracle Database 12c enables organizations to manage large, scalable, transactional, and data warehousing systems that deliver fast data access using low-cost modular storage.
- **Security:** Oracle Database 12c helps in protecting your information with unique secure configurations, data encryption and masking, and sophisticated auditing capabilities.
- **Information integration:** You can utilize Oracle Database 12c features to integrate data throughout the enterprise in a better way. You can also manage the changing data in your database by using Oracle Database 12c's advanced information lifecycle management capabilities.

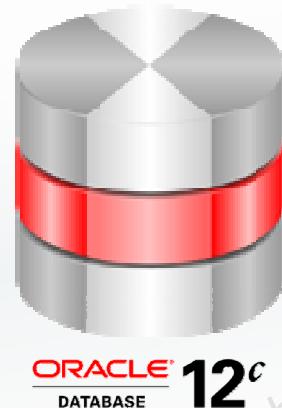
Lesson Agenda

- Course objectives
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources



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 © 2017, Oracle and/or its affiliates. All rights reserved.

Consider a situation where you put all your stationery in a single drawer without organizing. When you want a pencil immediately, you tend to search the entire drawer, which consumes a lot of time. This can be compared to the normal file storage in your system. But if the amount of information is huge, this system becomes inefficient.

Now imagine that you organize your stationery such that all the pencils/pens go into the first drawer, notepads in the second, glue sticks in third, and so on. Now when you want to fetch a pencil immediately, you will know where to find it efficiently. This can be compared to the relational model of storage.

Oracle Database is a Relational Database Management System (RDBMS). An RDBMS that implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an object relational database management system (ORDBMS). Oracle Database has extended the relational model to an object relational model, making it possible to store complex business models in a relational database.

Some of the advantages are:

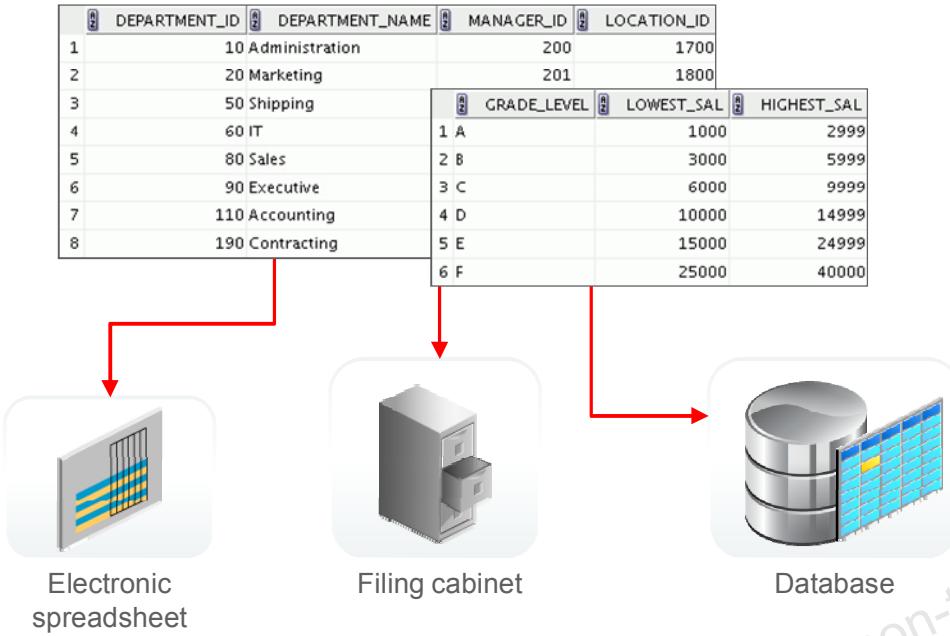
- Improved performance and functionality
- Better sharing of runtime data structures
- Larger buffer caches
- Deferrable constraints

Features such as parallel execution of insert, update, and delete operations; partitioning; and parallel-aware query optimization provide benefits to data warehouse applications.

The Oracle model supports client/server and web-based applications that are distributed and multi-tiered.

For more information about the relational and object relational model, refer to *Oracle Database Concepts for 12c Database*.

Data Storage on Different Media



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Relational Database Concept

- Dr. E. F. Codd proposed the relational model for database systems in 1970.
- It is the basis for 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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.

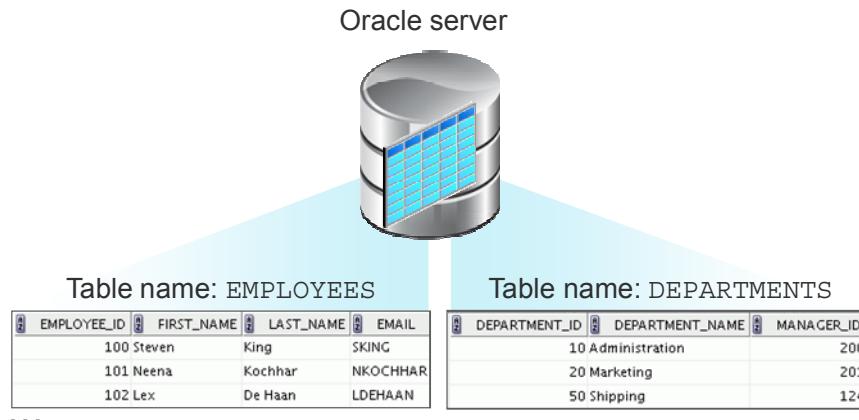
The common models used at that time were hierarchical and network, or even simple flat-file data structures. Relational database management systems (RDBMS) soon became very popular, especially for their ease of use and flexibility in structure. In addition, a number of innovative vendors, such as Oracle,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

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables controlled by the Oracle server.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Every organization has information that it must store and manage to meet its requirements.

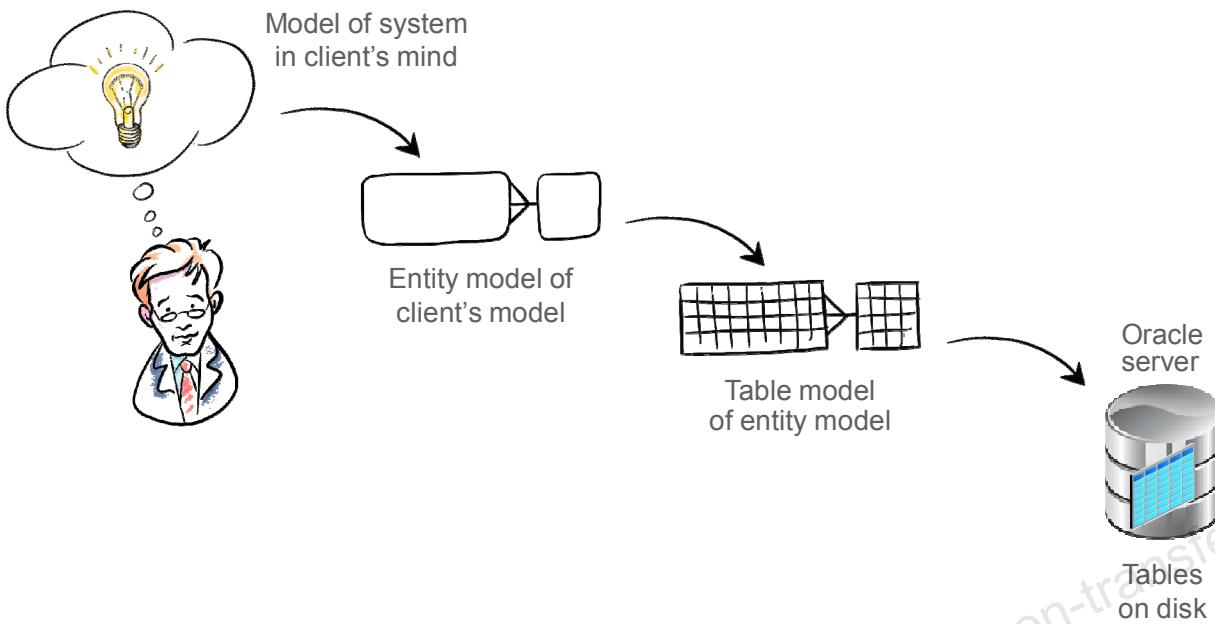
For example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table, and a salary table.

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

But before storing any information in the database, you need to first design a model of how and what data will be stored in the tables. In the following slide, you will learn about different data models that help in visualizing the architecture.



Data Models



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

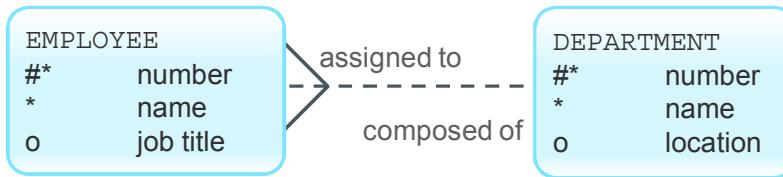
Models help you to communicate the concepts that are in people's minds. They can be used to do the following:

- 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. An Entity-Relationship model is one such data model.

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. . .”

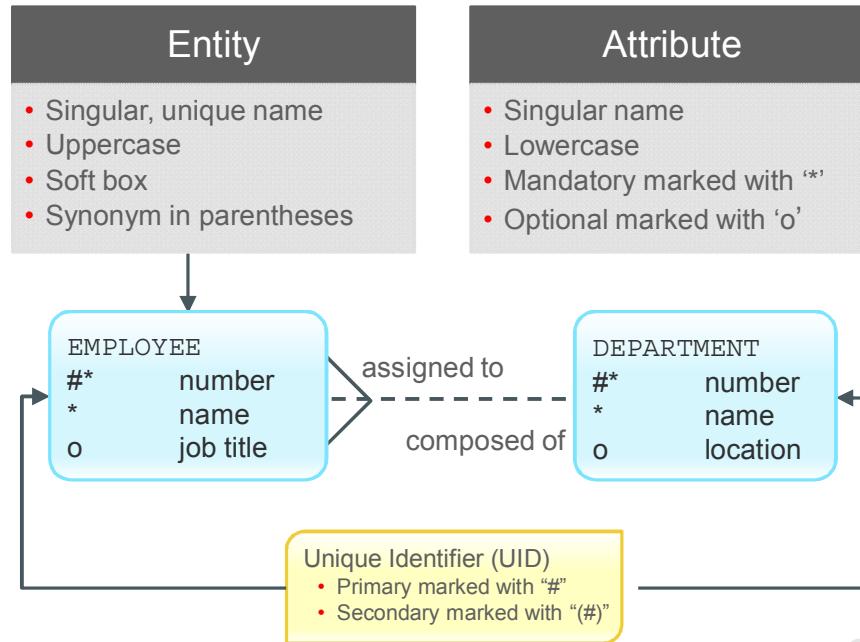
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Key Components

- **Entity:** An aspect of significance about which information must be known. For example, departments, employees, and orders are entities.
- **Attribute:** Something that describes or qualifies an entity. For example, in 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. For example, an employee assigned to a department is a relationship between the employee and department entities.

Entity Relationship Modeling Conventions



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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*

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”

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

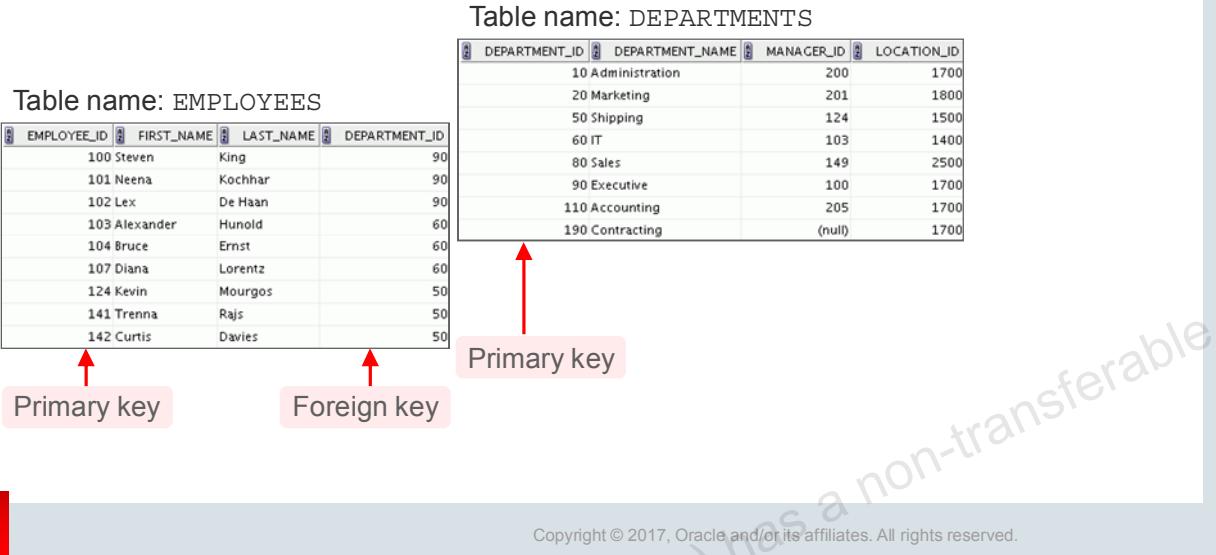
A unique identifier (UID) is any combination of attributes or relationships, or both, that serves to distinguish occurrences of an entity. Each entity occurrence must be uniquely identifiable.

- Tag each attribute that is part of the UID with a hash sign “#”.
- Tag secondary UIDs with a hash sign in parentheses (#).

For example, employee ID is an unique identifier in EMPLOYEE entity since no two employees can have the same employee ID.

Relating Multiple Tables

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



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 records are listed below. Each table can have one **primary key**, which in effect names the row and ensures no duplicate rows exist.

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 table by using **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 the data in another table to organize information in separate, manageable units. Employee data can be kept logically distinct from the department data by storing them in separate tables.

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 structure of the EMPLOYEES table with six numbered annotations:

- 1**: A red box around the entire table.
- 2**: A red box around the first row.
- 3**: A red box around the SALARY column.
- 4**: A red box around the DEPARTMENT_ID column.
- 5**: A red box around the last two rows.
- 6**: A red box around the last three columns.

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
145	Eleni	Zlotkey	10500	0.2	80
174	Ellen	Abel	11000	0.3	80
176	Jonathon	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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

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
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

Human Resources (HR) application

HR Application

Basic Search:

Advanced Search:

First Name Emp. ID
Last Name Department

GO

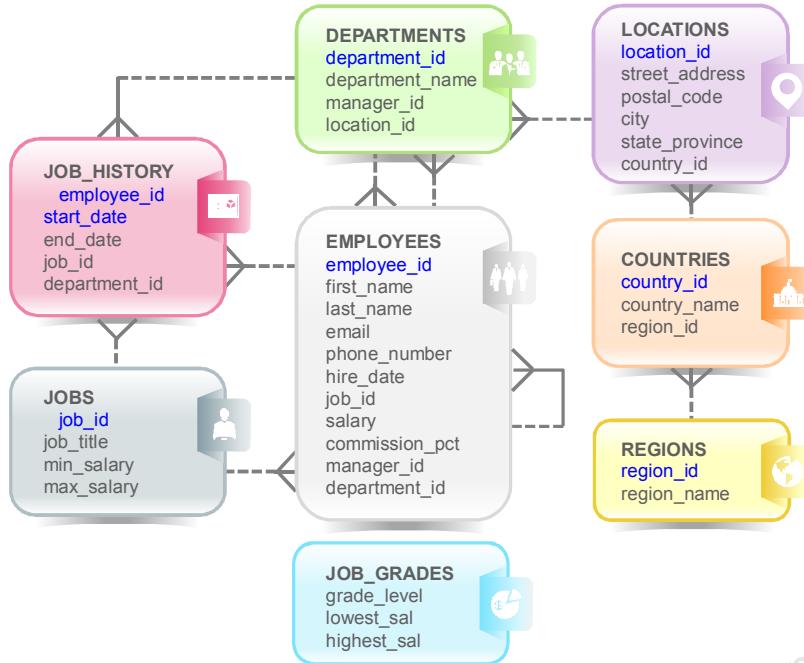


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The slide shows an example of a sample HR application, which is generally used by the HR department in the organization. The HR application consists of details of all the employees in the organization whose information is stored in various tables such as EMPLOYEES, DEPARTMENTS, JOBS, LOCATIONS, etc. The HR managers use this application to search for an employee's details, to update employee records such as when an employee gets promoted, to add new employee records such as when an employee joins the organization, or delete employee records of employees who have quit. In addition to these, the HR managers can use this application to generate a variety of reports, such as the number of employees who have quit and the number of new hires, average salaries in each department, list of employees who received a bonus this year, etc.

In this course, we will discuss various scenarios based on the sample HR application. You will also see different examples in the lessons based on the HR schema.

Tables Used in This Course



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

This course uses data from the following tables:

Table Descriptions

- The **EMPLOYEES** table contains information about all the employees, such as their first and last names, job IDs, salaries, hire dates, department IDs, and manager IDs. This table is a child of the **DEPARTMENTS** table.
- The **DEPARTMENTS** table contains information such as the department ID, department name, manager ID, and location ID. This table is the primary key table to the **EMPLOYEES** table.
- The **LOCATIONS** table contains department location information. It contains location ID, street address, city, state province, postal code, and country ID information. It is the primary key table to the **DEPARTMENTS** table and is a child of the **COUNTRIES** table.
- The **COUNTRIES** table contains the country names, country IDs, and region IDs. It is a child of the **REGIONS** table. This table is the primary key table to the **LOCATIONS** table.
- The **REGIONS** table contains region IDs and region names of the various countries. It is a primary key table to the **COUNTRIES** table.

- The `JOB_GRADES` table identifies a salary range per job grade. The salary ranges do not overlap.
- The `JOB_HISTORY` table stores job history of the employees.
- The `JOBS` table contains job titles and salary ranges.

Tables Used in the Course

EMPLOYEES

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-05	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	AC_MGR	12008
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-07	ST_MAN	5800
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-03	ST_CLERK	3500
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-05	ST_CLERK	3100
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-06	ST_CLERK	2600
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-06	ST_CLERK	2500
12	149	Eleni	Zlotkey	EZLOTEKY	011.44.1344.429018	29-JAN-08	SA_MAN	10500
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-04	SA REP	11000
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-06	SA REP	8600
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-07	SA REP	7000
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MKT_MAN	13000
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MKT REP	6000
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008
20	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-02	AC_ACCOUNT	8300

JOB_GRADES

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

DEPARTMENTS

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 A.

Lesson Agenda

- Course objectives
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources



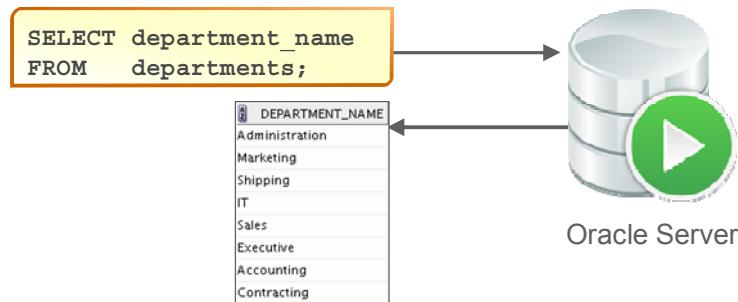
ORACLE®

Copyright © 2017, Oracle and/or its affiliates. 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 tables.)



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 also compliant to ISO Standard (SQL 1999).

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. Oracle Application Express is one such example.

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.

How SQL Works

- SQL is standalone and powerful.
- SQL processes groups of data.
- SQL lets you work with data at a logical level.



ORACLE

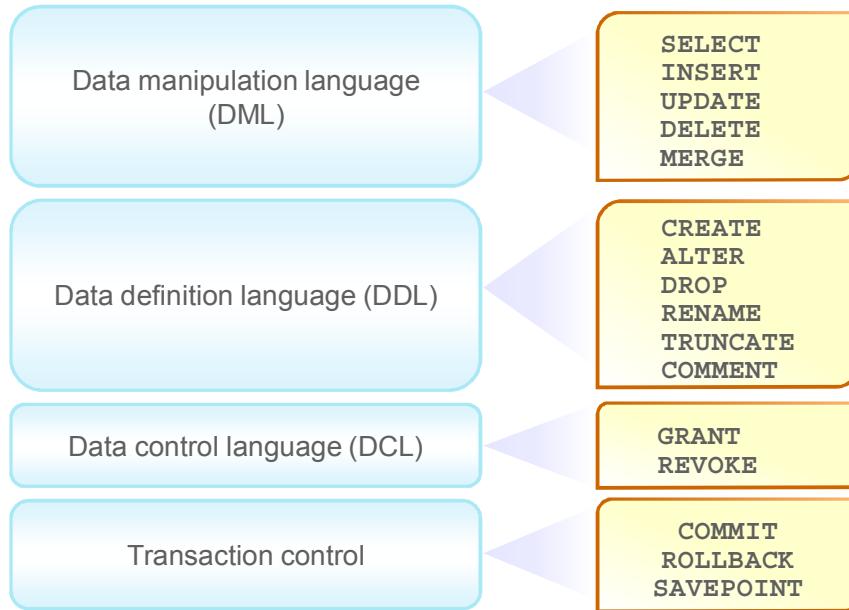
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using SQL benefits all types of users, including application programmers, database administrators, managers, and end users. The purpose of SQL is to provide an interface to a relational database such as Oracle Database. All SQL statements are instructions to the database. Some of the features of SQL are:

- It processes sets of data as groups rather than as individual units.
- It provides automatic navigation to the data.
- It uses statements that are complex and powerful individually, and are therefore standalone.

SQL lets you work with data at the logical level. For example, to retrieve a set of rows from a table, you define a condition used to filter the rows. All rows satisfying the condition are retrieved in a single step and can be passed as a unit to the user, to another SQL statement, or to an application. You need not deal with the rows one by one, nor do you have to worry about how they are physically stored or retrieved.

SQL Statements Used in the Course



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL statements are used to access the database and maintain the data.

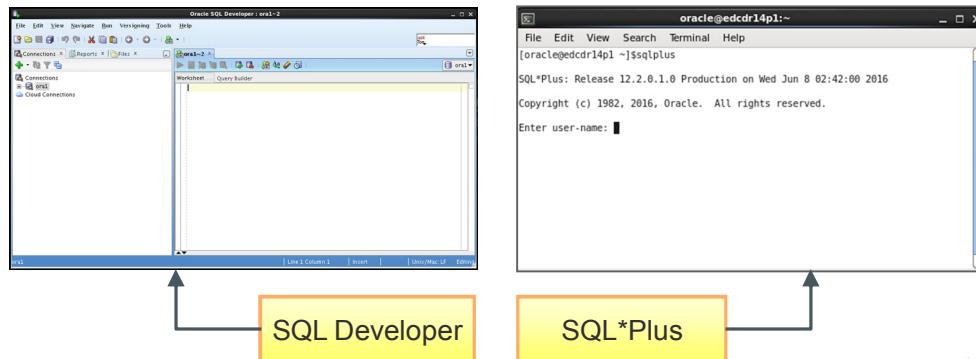
SQL statements supported by Oracle comply with industry standards. Oracle Corporation ensures future compliance with evolving standards by actively involving key personnel in SQL standards committees. The industry-accepted committees are ANSI and International Standards Organization (ISO). Both ANSI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

Statements	Description
SELECT INSERT UPDATE DELETE MERGE	Retrieve data from the database, enter new rows, change existing rows, and remove unwanted rows from tables in the database, respectively. Collectively known as <i>Data Manipulation Language</i> (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Set up, change, and remove data structures from tables. Collectively known as <i>Data Definition Language</i> (DDL)
GRANT REVOKE	Provide or remove access rights to both the Oracle Database and the structures within it.
COMMIT ROLLBACK SAVEPOINT	Manage 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.
- The SQL*Plus command-line interface can also be used.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

There are various development environments for writing SQL statements. Oracle SQL Developer and Oracle SQL*Plus are the commonly used tools.

Oracle SQL Developer

Oracle SQL Developer is a graphical interface for developing SQL. It provides features to view the database components and update values without writing any SQL query. In this course, Oracle SQL Developer is used to create and execute example SQL statements. You will use Oracle SQL Developer to perform the hands-on exercises.

Oracle SQL*Plus

If you like working with a command-line interface, you can use Oracle SQL*Plus, which is a command-line based environment. It can also be used to run all SQL commands covered in this course.

Note

- See Appendix B for information about using Oracle SQL Developer, including simple instructions on the installation process.
- See Appendix C for information about using Oracle SQL*Plus.

Introduction to Oracle Live SQL

- Easy way to learn, access, test, and share SQL and PL/SQL scripts on Oracle Database
- Sign up and use it free of cost.



Oracle Live SQL is another environment where you can write and execute SQL statements.

You can now learn SQL without the need to install a database or download any tool. Oracle Live SQL exists to provide the Oracle database community with an easy online way to test and share SQL and PL/SQL application development concepts.

Let us look at some of the features of Oracle Live SQL:

- Provides browser-based SQL worksheet access to an Oracle database schema
- Has the ability to save and share SQL scripts
- Provides a schema browser to view and extend database objects
- Provides access to interactive educational tutorials
- Provides customized data access examples for PL/SQL, Java, PHP, C

You can continue to learn SQL by using Live SQL yourself. All you need is your Oracle Technology Network (OTN) credentials and an interest in learning SQL.

Note: Oracle Live SQL cannot be used to execute the lab exercises without the initial schema setup.

Lesson Agenda

- Course objectives
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

What is Oracle SQL Developer?

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



SQL Developer

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

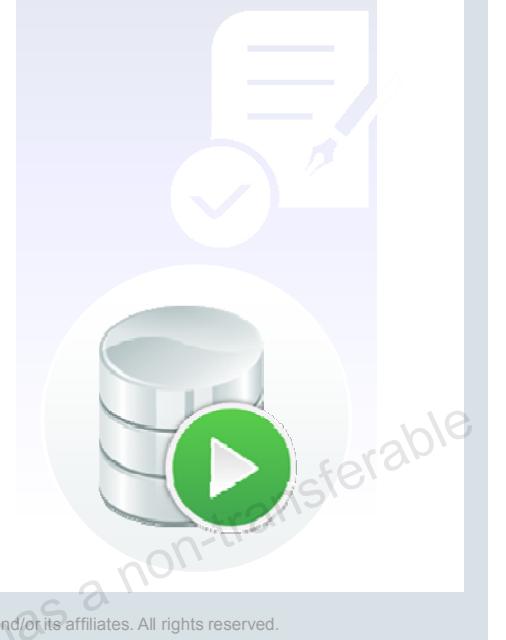
- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports

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

SQL Developer is the interface to administer the Oracle Application Express Listener. The new interface enables you to specify global settings and multiple database settings with different database connections for the Application Express Listener. SQL Developer provides the option to drag and drop objects by table or column name onto the worksheet. It provides improved DB Diff comparison options, GRANT statement support in the SQL editor, and DB Doc reporting. Additionally, SQL Developer includes support for Oracle Database 12c features.

Specifications of SQL Developer

- Is shipped along with Oracle Database 12c Release 1
- Is developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity using the JDBC Thin driver
- Connects to Oracle Database version 9.2.0.1 and later



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Oracle SQL Developer is shipped along with Oracle Database 12c Release 1 by default. SQL Developer is developed in Java, leveraging the Oracle JDeveloper integrated development environment (IDE). Therefore, it is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms.

The default connectivity to the database is through the Java Database Connectivity (JDBC) Thin driver and, therefore, no Oracle Home is required. SQL Developer does not require an installer and you need to simply unzip the downloaded file. With SQL Developer, users can connect to Oracle Databases 9.2.0.1 and, later, and all Oracle database editions, including Express Edition.

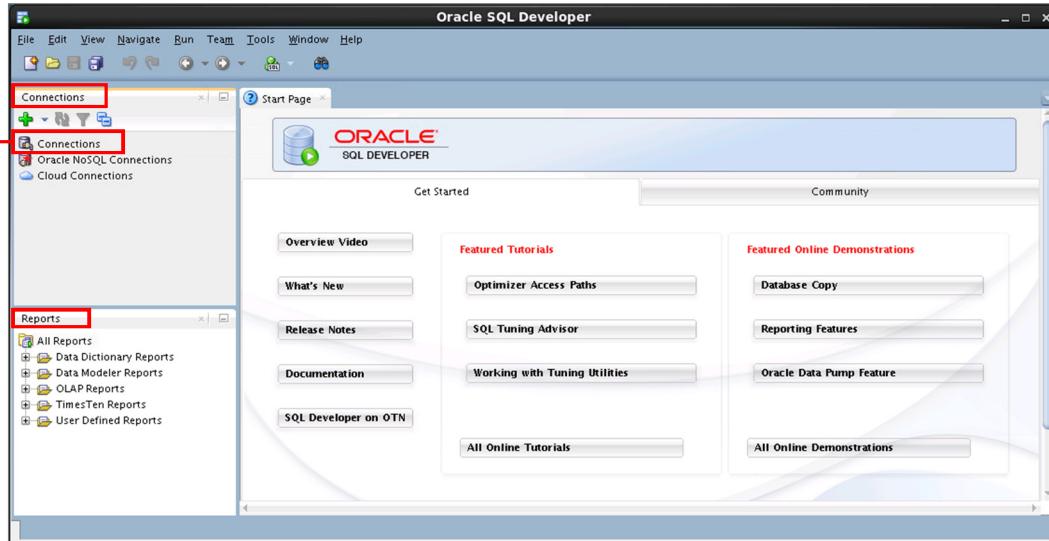
Note: For Oracle Database 12c Release 1, you have to download and install SQL Developer. SQL Developer is freely downloadable from the following link:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>

For instructions on how to install SQL Developer, see the website at:

<http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>

SQL Developer 3.2 Interface



You must define a connection to start using SQL Developer for running SQL queries on a database schema.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The SQL Developer interface contains two main navigation tabs:

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

General Navigation and Use

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

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

Menus

The following menus contain standard entries, plus entries for features that are specific to SQL Developer:

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options
- **Versioning:** Provides integrated support for the following versioning and source control systems—Concurrent Versions System (CVS) and Subversion
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet. It also contains options related to migrating third-party databases to Oracle.

Note: The Run menu also contains options that are relevant when a function or procedure is selected for debugging.

Creating a Database Connection

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

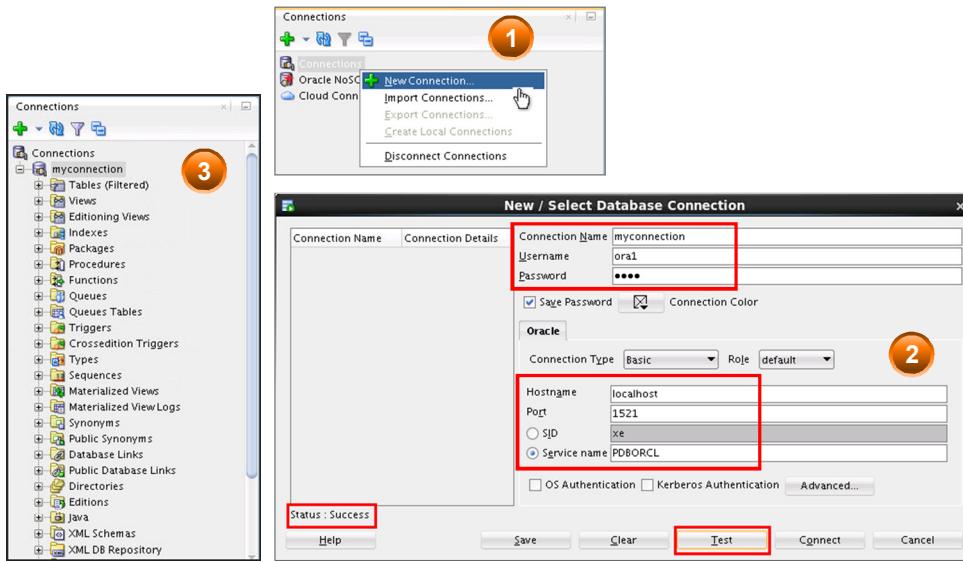
By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory, but it can also be in the directory specified by the `TNS_ADMIN` environment variable or registry value. When you start SQL Developer and open the Database Connections dialog box, SQL Developer automatically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

Note: On Windows, if the `tnsnames.ora` file exists, but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

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

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

Creating a Database Connection



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New / Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
 - a. From the Role drop-down list, you can select either *default* or SYSDBA. (You choose SYSDBA for the `sys` user or any user with database administrator privileges.)
 - b. You can select the connection type as:
 - Basic:** In this type, enter host name and SID for the database that you want to connect to. Port is already set to 1521. You can also choose to enter the Service name directly if you use a remote database connection.
 - TNS:** You can select any one of the database aliases imported from the `tnsnames.ora` file.
 - LDAP:** You can look up database services in Oracle Internet Directory, which is a component of Oracle Identity Management.
 - Advanced:** You can define a custom JDBC URL to connect to the database.
 - Local/Bequeath:** If the client and database exist on the same computer, a client connection can be passed directly to a dedicated server process without going through the listener.
 - c. Click Test to ensure that the connection has been set correctly.
 - d. Click Connect.

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

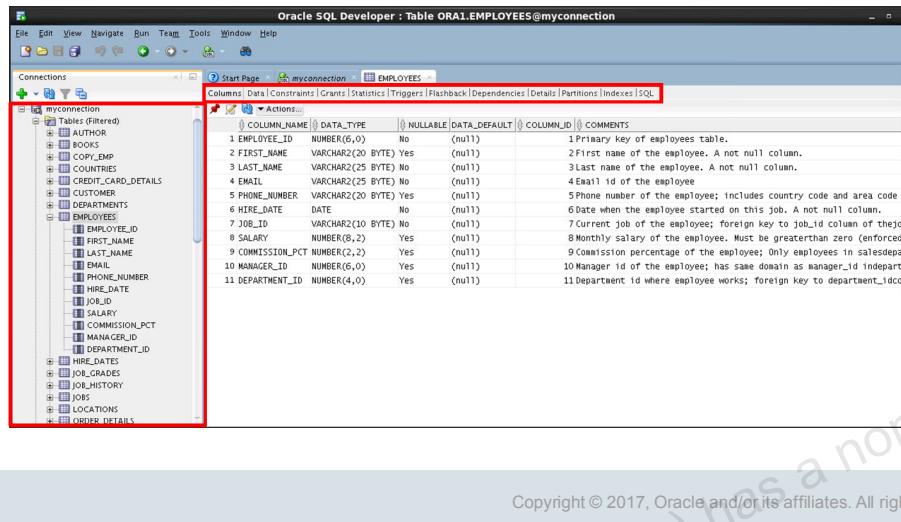
3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions (dependencies, details, statistics, and so on).

Note: From the same New>Select Database Connection window, you can define connections to non-Oracle data sources using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to:

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

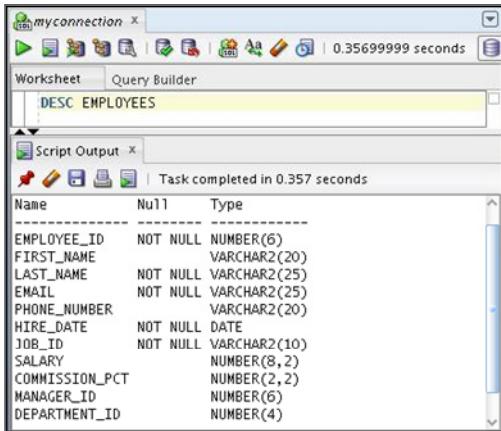
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

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

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

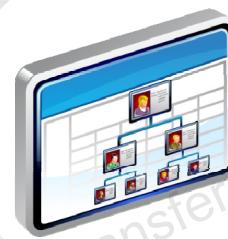
Displaying the Table Structure

Use the DESCRIBE command to display the structure of a table:



The screenshot shows the Oracle SQL Developer interface. In the top-left corner, there's a window titled "myconnection x". Below it, a "Worksheet" tab is active, showing the command "DESC EMPLOYEES". To the right of the worksheet is a "Script Output" tab which displays the results of the DESCRIBE command. The output is a table with three columns: "Name", "Null", and "Type". The data rows are as follows:

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)

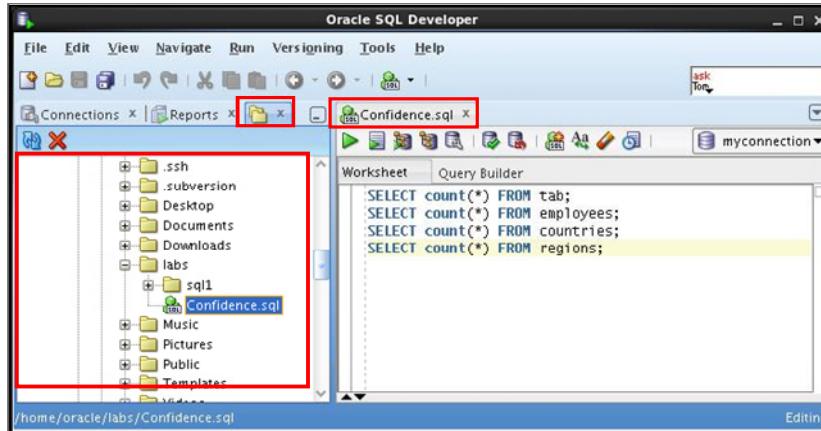
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In SQL Developer, you can also display the structure of a table using the DESCRIBE command. The result of the command is a display of column names and data types as well as an indication of whether a column must contain data.

Browsing Files

Use the File Navigator to explore the file system and open system files.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



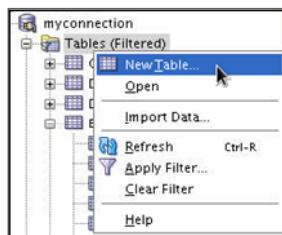
Browsing Database Objects

You can use the File Navigator to browse and open system files.

- To view the File Navigator, click the View tab and select Files, or select View > Files.
- To view the contents of a file, double-click a file name to display its contents in the SQL Worksheet area.

Creating a Schema Object

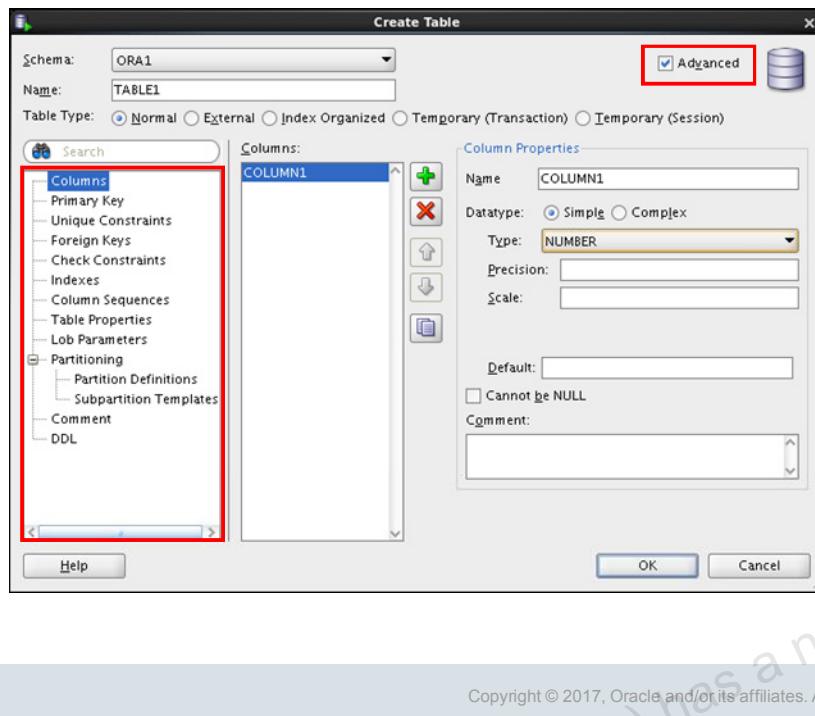
- SQL Developer supports the creation of any schema object by:
 - Executing a SQL statement in SQL Worksheet
 - Using the context menu
- Edit the objects by using an edit dialog box or one of the many context-sensitive menus.
- View the DDL for adjustments such as creating a new object or editing an existing schema object.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Creating a New Table: Example



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the Create Table dialog box, if you do not select the Advanced check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the Advanced check box, the Create Table dialog box changes to one with multiple options, in which you can specify an extended set of features while you create the table.

The example in the slide shows how to create the DEPENDENTS table by selecting the Advanced check box.

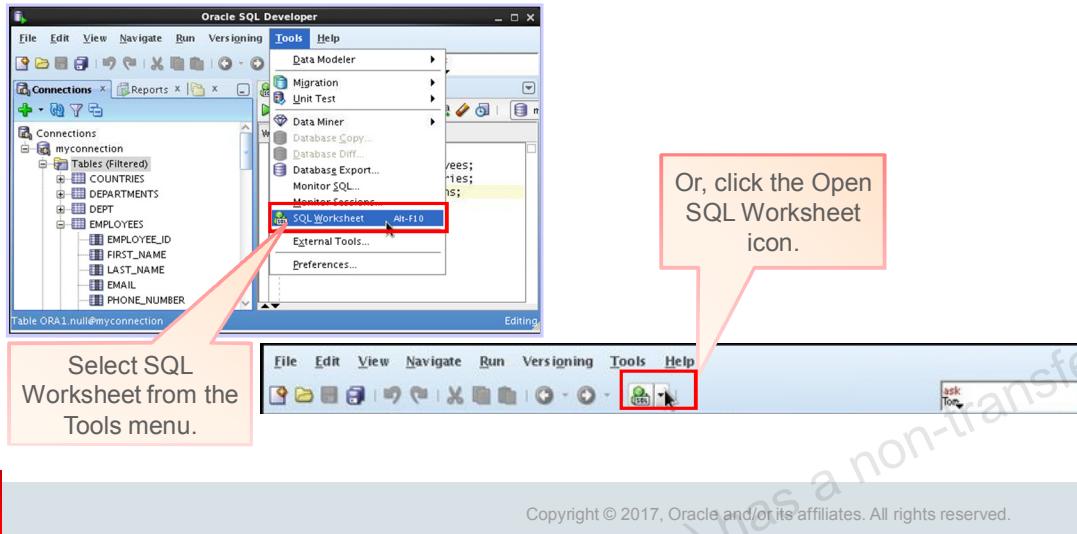
To create a new table, perform the following steps:

1. In the Connections Navigator, right-click Tables and select Create TABLE.
2. In the Create Table dialog box, select Advanced.
3. Specify the column information.
4. Click OK.

Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

Using SQL Worksheet

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. SQL Worksheet supports SQL*Plus statements to a certain extent. SQL*Plus statements that are not supported by SQL Worksheet are ignored and not passed to the database.

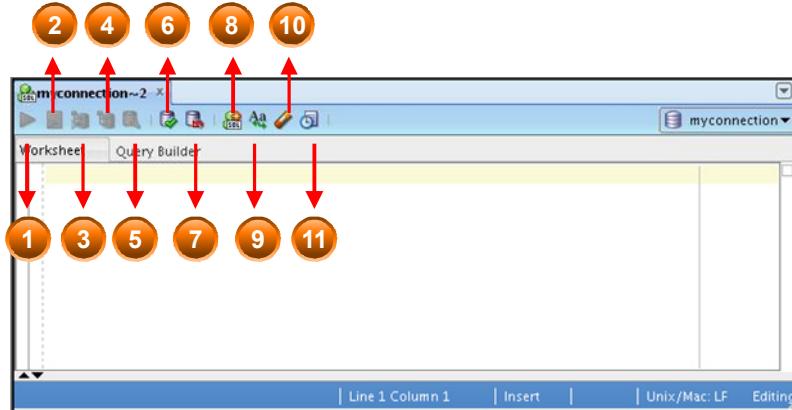
You can specify the actions that can be processed by the database connection associated with the worksheet, such as:

- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL Worksheet by using one of the following:

- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Using SQL Worksheet



The ORACLE logo in white text on a red background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

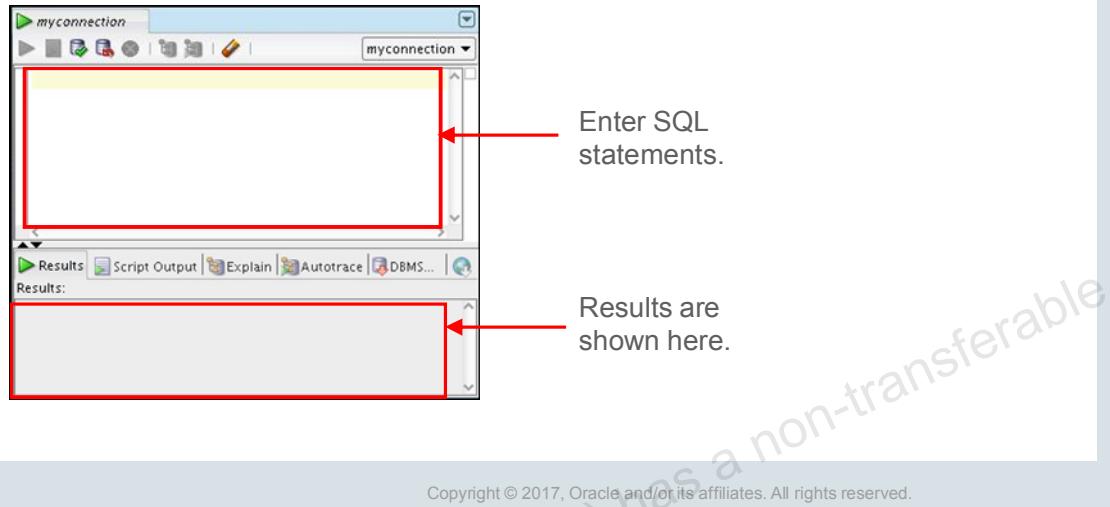
You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of the SQL statements that you have executed. You can use SQL Worksheet toolbar that contains icons to perform the following tasks:

1. **Run Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all the statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Autotrace:** Generates trace information for the statement
4. **Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
5. **SQL Tuning Advisory:** Analyzes high-volume SQL statements and offers tuning recommendations
6. **Commit:** Writes any changes to the database and ends the transaction
7. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction

8. **Unshared SQL Worksheet:** Creates a separate unshared SQL Worksheet for a connection
9. **To Upper/Lower/InitCap:** Changes the selected text to uppercase, lowercase, or initcap, respectively
10. **Clear:** Erases the statement or statements in the Enter SQL Statement box
11. **SQL History:** Displays a dialog box with information about the SQL statements that you have executed

Using SQL Worksheet

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

Executing SQL Statements

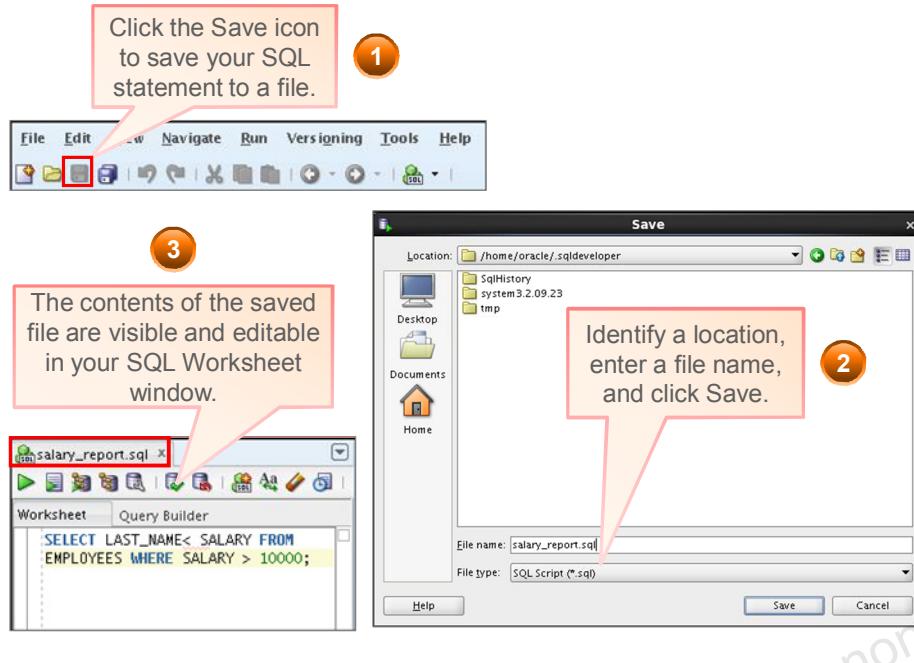
Use the Enter SQL Statement box to enter single or multiple SQL statements.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Saving SQL Scripts



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can save your SQL statements from SQL Worksheet to a text file. To save the contents of the Enter SQL Statement box, perform the following steps:

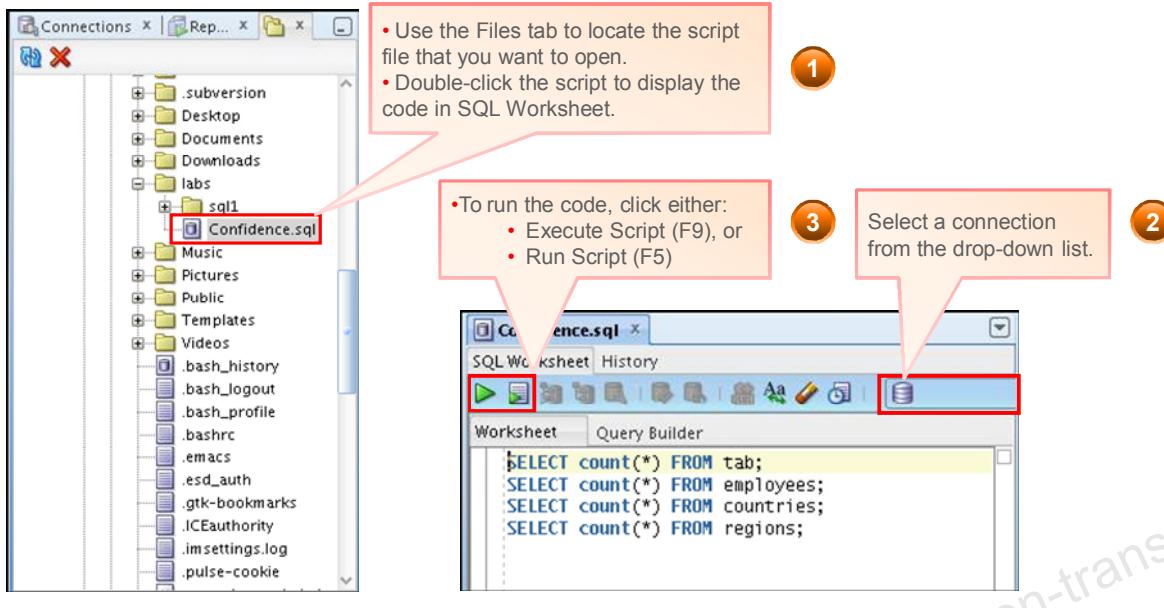
1. Click the Save icon or use the File > Save menu item.
2. In the Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file is displayed as a tabbed page.

Script Pathing

You can select a default path to look for scripts and to save scripts. Under Tools > Preferences > Database > Worksheet Parameters, enter a value in the "Select default path to look for scripts" field.

Executing Saved Script Files: Method 1



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

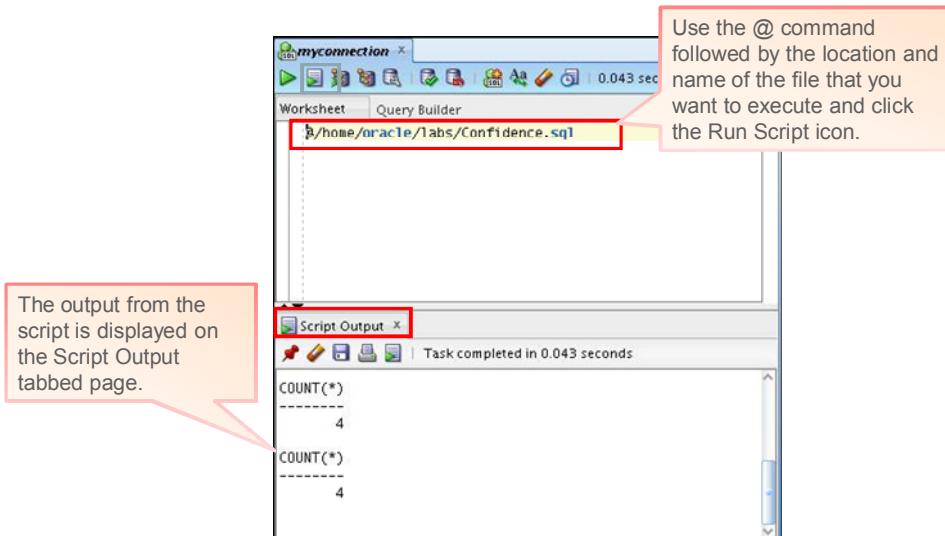
To open a script file and display the code in the SQL Worksheet area, perform the following steps:

1. In the files navigator, select (or navigate to) the script file that you want to open.
2. Double-click the file to open it. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the connection drop-down list.
4. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection that you want to use for the script execution.

Alternatively, you can also do the following:

1. Select File > Open. The Open dialog box is displayed.
2. In the Open dialog box, select (or navigate to) the script file that you want to open.
3. Click Open. The code of the script file is displayed in the SQL Worksheet area.
4. Select a connection from the connection drop-down list.
5. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection that you want to use for the script execution.

Executing Saved Script Files: Method 2



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To run a saved SQL script, perform the following steps:

1. Use the @ command followed by the location and the name of the file that you want to run in the Enter SQL Statement window.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The File Save dialog box appears and you can identify a name and location for your file.

Formatting the SQL Code

The screenshot shows the Oracle SQL Developer interface. On the left, there is a code editor window containing SQL code. A red callout box labeled "Before formatting" points to the first few lines of the code. On the right, a context menu is open over the code editor, with a red callout box labeled "After formatting" pointing to the "Format" option under the "Refactoring" section. The menu also includes options like "Advanced Format...", "Code Template", "Popup Describe", and "Open Declaration". Below the code editor, the formatted SQL code is displayed, showing correctly capitalized keywords and properly indented statements.

```
select employee_id, first_name, salary from employees e, departments d
where e.department_id=d.department_id and e.salary>3000;
```

```
SELECT employee_id,
       first_name,
       salary
  FROM employees e,
       departments d
 WHERE e.department_id=d.department_id
   AND e.salary>3000;
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

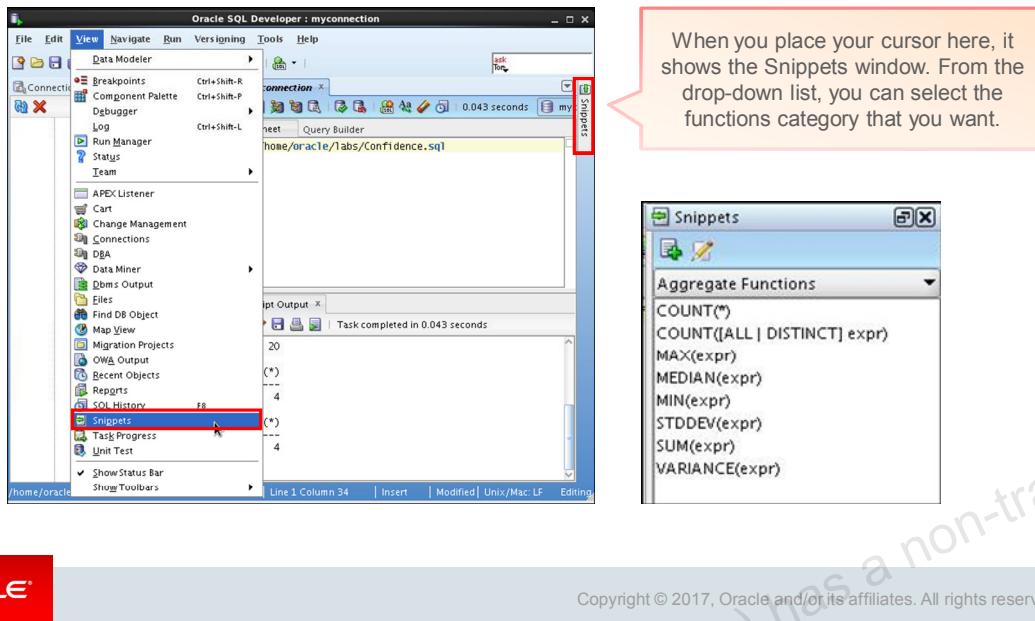
You may want to format the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has a feature for formatting the SQL code.

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

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

Using Snippets

Snippets are code fragments that may be just syntax or examples.



ORACLE

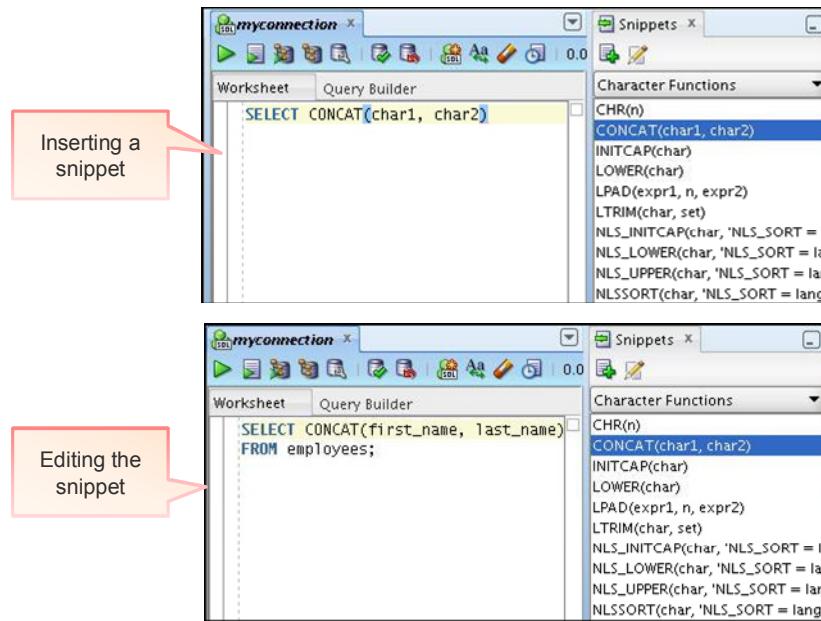
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You may want to use certain code fragments when you use SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has a feature called Snippets. Snippets are code fragments such as SQL functions, optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets to the Editor window.

To display Snippets, select View > Snippets.

The Snippets window is displayed on the right. You can use the drop-down list to select a group. A Snippets button is placed in the right window margin, so that you can display the Snippets window if it becomes hidden.

Using Snippets: Example



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

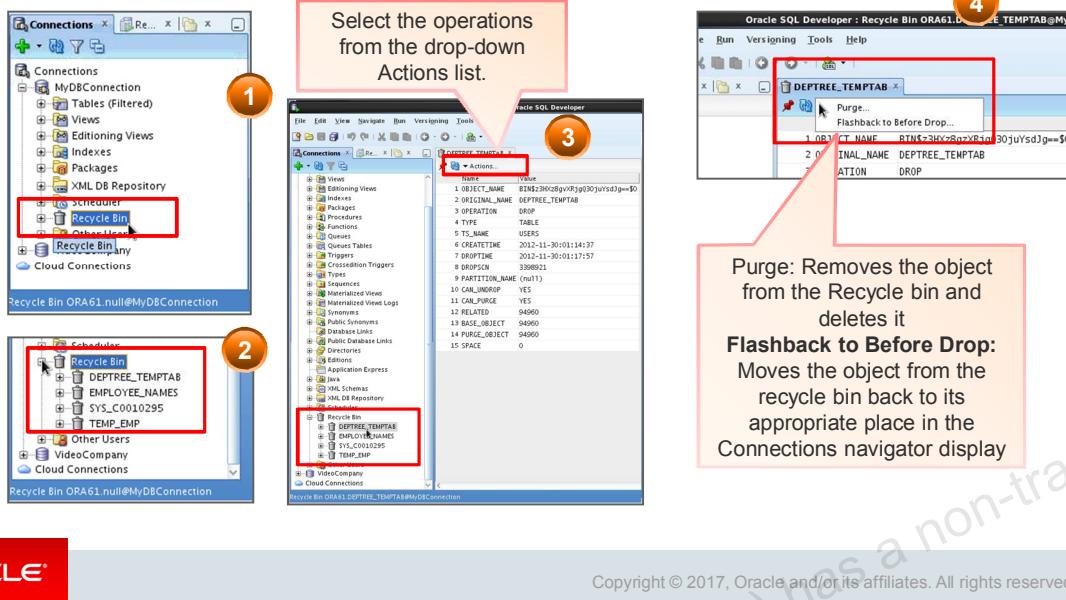
To insert a Snippet into your code in SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window to the desired place in your code. Then you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a SQL function in a tool tip, place the cursor over the function name.

The example in the slide shows that `CONCAT (char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited and the rest of the statement is added as in the following:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

Using the Recycle Bin

The recycle bin holds objects that have been dropped.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

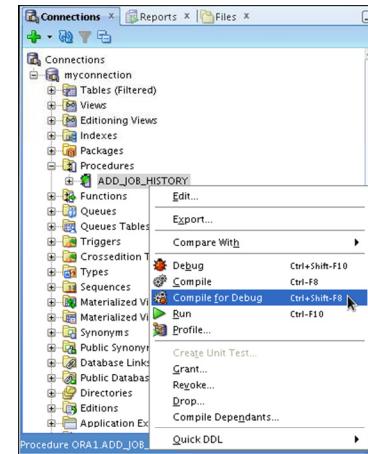
The recycle bin is a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and the likes are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

To use the recycle bin, perform the following steps:

- 1 In the Connections navigator, select (or navigate to) the recycle bin.
- 2 Expand Recycle Bin and click the object name. The object details are displayed in the SQL Worksheet area.
- 3 Click the Actions drop-down list and select the operation that you want to perform on the object.

Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the Compile for Debug option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use the Debug menu options to set breakpoints, and to perform Step Into and Step Over tasks.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In SQL Developer, you can debug PL/SQL procedures and functions. Using the Debug menu options, you can perform the following debugging tasks:

- **Find Execution Point** goes to the next execution point.
- **Resume** continues execution.
- **Step Over** bypasses the next method and goes to the next statement after the method.
- **Step Into** goes to the first statement in the next method.
- **Step Out** leaves the current method and goes to the next statement.
- **Step to End of Method** goes to the last statement of the current method.
- **Pause** halts execution, but does not exit, thus allowing you to resume execution.
- **Terminate** halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the function or procedure, click the Run or Debug icon on the Source tab toolbar.
- **Garbage Collection** removes invalid objects from the cache in favor of more frequently accessed and more valid objects.

These options are also available as icons on the Debugging tab of the output window.

Database Reporting

SQL Developer provides a number of predefined reports about the database and its objects.

The screenshot shows the Oracle SQL Developer interface with the 'Reports' tab selected. On the left, there's a tree view of report categories: All Reports, Data Dictionary Reports, All Objects, Collection Types, Dependencies, Invalid Objects, Object Count by Type, Object Distribution, Public Database Links, Public Synonyms, Application Express, ASH and AWR, Database Administration, Data Dictionary, PLSQL, Scheduler, Security, Streams, Table, XML, Data Modeler Reports, and User Defined Reports. On the right, a large table displays database objects. The columns are: #, Owner, Name, Type, Referenced_Owner, Referenced_Name, and Referenced_. The table lists numerous objects from the APEX_040100 schema, such as procedures, functions, tables, and packages, many of which are related to the MM_FLOW schema.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

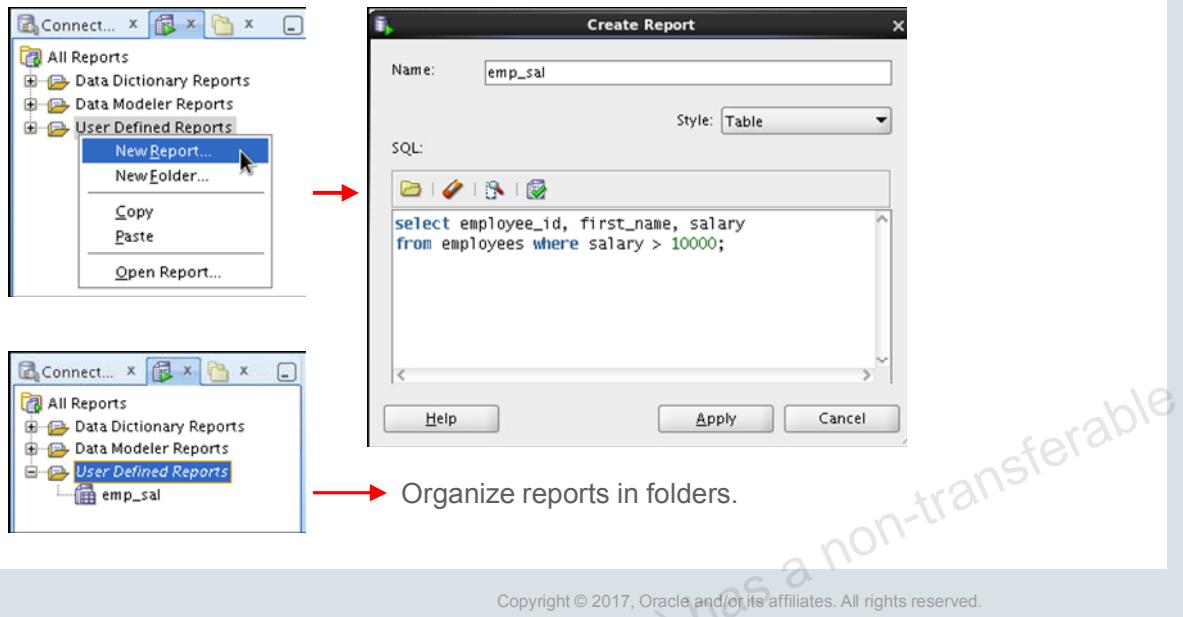
SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

To display reports, click the Reports tab on the left of the window. Individual reports are displayed in tabbed panes on the right of the window; for each report, you can select (by using a drop-down list) the database connection for which to display the report. For reports about objects, the objects shown are only those visible to the database user associated with the selected database connection, and the rows are usually ordered by Owner. You can also create your own user-defined reports.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



User-defined reports are reports created by SQL Developer users. To create a user-defined report, perform the following steps:

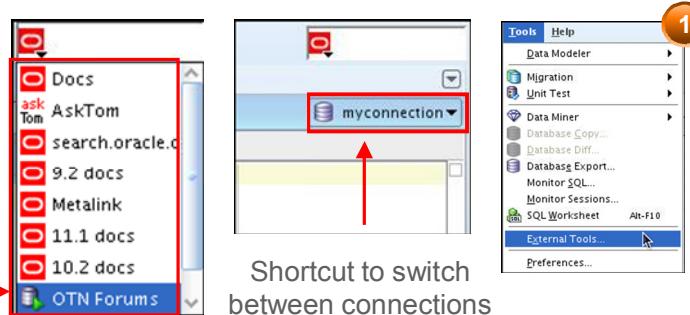
1. Right-click the User Defined Reports node under Reports and select Add Report.
2. In the Create Report dialog box, specify the report name and the SQL query to retrieve information for the report. Then click Apply.

In the example in the slide, the report name is specified as `emp_sal`. An optional description is provided indicating that the report contains details of employees with `salary >= 10000`. The complete SQL statement for retrieving the information to be displayed in the user-defined report is specified in the SQL box. You can also include an optional tool tip to be displayed when the cursor stays briefly over the report name in the Reports navigator display.

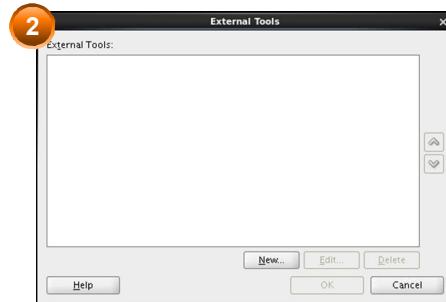
You can organize user-defined reports in folders and you can create a hierarchy of folders and subfolders. To create a folder for user-defined reports, right-click the User Defined Reports node or any folder name under that node and select Add Folder. Information about user-defined reports, including any folders for these reports, is stored in a file named `UserReports.xml` in the directory for user-specific information.

Search Engines and External Tools

Links to popular search engines and discussion forums



Shortcut to switch between connections



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

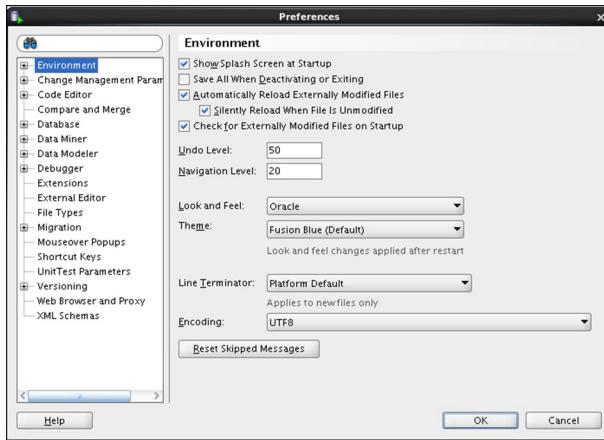
To enhance the productivity of developers, SQL Developer has added quick links to popular search engines and discussion forums such as AskTom, Google, and so on. Also, you have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver, available to you.

You can add external tools to the existing list or even delete shortcuts to the tools that you do not use frequently. To do so, perform the following steps:

1. From the Tools menu, select External Tools.
2. In the External Tools dialog box, select New to add new tools. Select Delete to remove any tool from the list.

Setting Preferences

- Customize the SQL Developer interface and environment.
- In the Tools menu, select Preferences.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

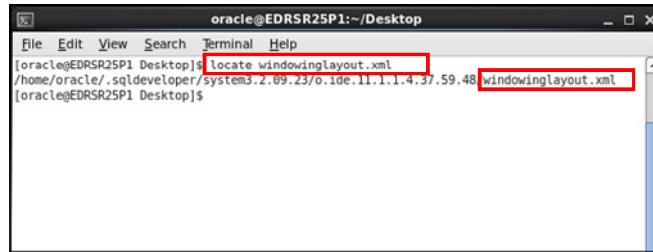
You can customize many aspects of the SQL Developer interface and environment by modifying SQL Developer preferences according to your needs. To modify SQL Developer preferences, select Tools, and then Preferences.

The preferences are grouped into the following categories:

- Environment
- Change Management Parameter
- Code Editors
- Compare and Merge
- Database
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types

- Migration
- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

Resetting the SQL Developer Layout



```
oracle@EDRSR25P1:~/Desktop
File Edit View Search Terminal Help
[oracle@EDRSR25P1 Desktop] locate windowlayout.xml
/home/oracle/.sqldeveloper/system3.2.09.23/o.ide.11.1.1.4.37.59.48/windowlayout.xml
[oracle@EDRSR25P1 Desktop]$
```



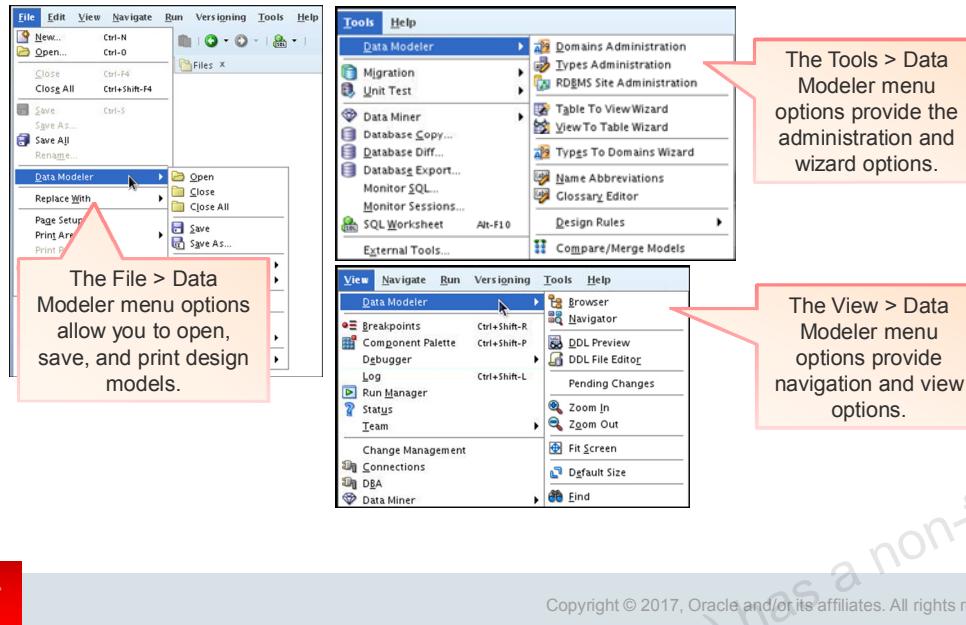
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

While working with SQL Developer, if the Connections Navigator disappears or if you cannot dock the Log window in its original place, perform the following steps to fix the problem:

1. Exit SQL Developer.
2. Open a terminal window and use the locate command to find the location of `windowlayout.xml`.
3. Go to the directory that has `windowlayout.xml` and delete it.
4. Restart SQL Developer.

Data Modeler in SQL Developer

SQL Developer includes an integrated version of SQL Developer Data Modeler.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the integrated version of the SQL Developer Data Modeler, you can:

- Create, open, import, and save a database design
- Create, modify, and delete Data Modeler objects

To display Data Modeler in a pane, click Tools, and then Data Modeler. The Data Modeler menu under Tools includes additional commands, for example, that enable you to specify design rules and preferences.

Lesson Agenda

- Course objectives
- Overview of Oracle Database 12c and related products
- Overview of relational database management concepts and terminologies
- Human Resource (HR) Schema and the tables used in this course
- Introduction to SQL and its development environments
- Introduction to SQL Developer
- Oracle Database 12c SQL Documentation and Additional Resources

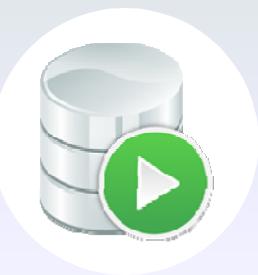


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Oracle Database Documentation

- *Oracle Database New Features Guide*
- *Oracle Database Reference*
- *Oracle Database SQL Language Reference*
- *Oracle Database Concepts*
- *Oracle Database SQL Developer User's Guide*



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Navigate to <https://docs.oracle.com/en/database/database.html> to access the Oracle Database 12c documentation library.

Additional Resources

For additional information about Oracle Database 12c, refer to the following:

- *Oracle Database 12c: New Features eStudies*
- *Oracle Learning Library*:
 - <http://www.oracle.com/goto/oll>
- The online SQL Developer Home Page, which is available at:
 - http://www.oracle.com/technology/products/database/sql_developer/index.html
- The SQL Developer tutorial, which is available online at:
 - <http://download.oracle.com/oll/tutorials/SQLDeveloper/index.htm>



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned about:

- The goals of the course
- Features of Oracle Database 12c
- The theoretical and physical aspects of a relational database
- Oracle server's implementation of RDBMS and ORDBMS
- The development environments that can be used for this course
- The database and schema used in this course
- Use SQL Developer
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports
- Browse the Data Modeling options in SQL Developer



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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, which you use to store and manage information by using SQL
- Oracle Fusion Middleware, which you use to 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 1: Overview

This practice covers the following topics:

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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/sql11/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.

Retrieving Data Using the SQL SELECT Statement

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cornelia Sirbu (vrabie.cornelia@gmail.com) has a non-transferable
license to use this Student Guide.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

To extract data from a 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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Unauthorized reproduction or distribution prohibited. Copyright© 2014, Oracle and/or its affiliates.

HR Application Scenario

Alex, an HR manager in India, wants a report of all the employees in the Accounting department of the organization. The HR application consists of a database of all the employees in the organization. So, how does Alex search based on the criteria in the HR application?

Alex finds it very efficient to use the HR application to generate reports. He logs in to the application, enters 'Accounting' in the Department field, and clicks Go.

The HR application fires a SQL SELECT statement to query the database for all employees in the Accounting department. Alex then sees the result on his application.

Basically, you can query the database for information by writing conditional and complex SQL statements. In this lesson, you will learn more about SQL SELECT statements.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Basic SELECT Statement

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

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

Selecting from a table →



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Use the **SELECT** statement to retrieve data from one or more tables or views in the database.

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
<i>column / expression</i>	Selects the named column or the expression
<i>alias</i>	Gives different headings to the selected columns
FROM <i>table</i>	Specifies the table containing the columns

Throughout this course, you will see that 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Selecting Specific Columns

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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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;
```

Selecting from DUAL

- DUAL is a table automatically created by Oracle Database.
- DUAL has one column called DUMMY, of data type VARCHAR (1) , and contains one row with a value x.

```
SELECT *
FROM dual;
```

DUMMY
1 X

```
SELECT sysdate
FROM dual;
```

SYSDATE
1 14-JUN-16



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you install Oracle Database, a DUAL table is automatically created. This table is in the SYS user schema but is accessible by the name DUAL to all users. When you display the contents of the DUAL table, you will observe that it has one column, DUMMY, defined to be varchar(1) , and contains one row with a value x.

Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Because DUAL has only one row, the constant is returned only once. Alternatively, you can select a constant, pseudocolumn, or expression from any table, but the value will be returned as many times as there are rows in the table.

For example, if you want to compute the expression $12*4 / 5 + 5 * 8$, use the following statement:

```
select 12*4/5+5*8
from dual;
```

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 © 2017, Oracle and/or its affiliates. All rights reserved.

Writing SQL Statements

By using the following simple rules and guidelines, you can construct valid statements that are 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 column 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 cursor 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 © 2017, Oracle and/or its affiliates. All rights reserved.

In SQL Developer, column headings are displayed in uppercase and are left-aligned.

Run the following SQL statement and observe the column headings in the output:

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

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

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Arithmetic Expressions

You can create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using Arithmetic Operators

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

#	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Remember that the Oracle server ignores blank spaces before and after the arithmetic operator.

Rules of Precedence

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

Operator Precedence

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

1

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100
4	Hunold	9000	108100

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

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200
4	Hunold	9000	109200



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 $(12 * \text{salary}) + 100$ with no change in the result.

Using Parentheses

You can override the rules of precedence by using parentheses to specify the desired order in which 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
King	AD_PRES	24000	(null)
Kochhar	AD_VP	17000	(null)
De Haan	AD_VP	17000	(null)
...			
12 Zlotkey	SA_MAN	10500	0.2
13 Abel	SA_REP	11000	0.3
14 Taylor	SA_REP	8600	0.2
15 Grant	SA_REP	7000	0.15
...			
18 Fay	MK_REP	6000	(null)
19 Higgins	AC_MGR	12008	(null)
20 Gietz	AC_ACCOUNT	8300	(null)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

You can select columns with **NULL** value in a **SELECT** query and **NULL** values can be part of an arithmetic expression. Any arithmetic expression using **NULL** values results into **NULL**.

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

You can see that 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 and on the right pane, for “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
King	(null)
Kochhar	(null)
De Haan	(null)

12 Zlotkey	25200
13 Abel	39600
14 Taylor	20640
15 Grant	12600

17 Hartstein	(null)
18 Fay	(null)
19 Higgins	(null)
20 Gietz	(null)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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 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	King	(null)
2	Kochhar	(null)
3	De Haan	(null)
4	Hunold	(null)

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

#	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000
4	Hunold	108000



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Concatenation Operator

The 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	Abe1SA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP
6	GietzAC_ACCOUNT
7	GrantSA_REP
8	HartsteinMK_MAN

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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
11	King is a AD_PRES
...	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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;
```

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:



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Duplicate Rows

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

1

```
SELECT department_id  
FROM employees;
```

#	DEPARTMENT_ID
1	90
2	90
3	90
4	60
5	60
6	60
7	50
8	50

...

2

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

#	DEPARTMENT_ID
1	(null)
2	90
3	20
4	110
5	50
6	80
7	60
8	10



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Displaying Table Structure

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

The screenshot shows the Oracle SQL Developer interface. At the top, a yellow bar displays the command: `DESC [RIBE] tablename`. Below this, the Connections tree shows a connection named "myconnection" with tables "COUNTRIES" and "DEPARTMENTS" selected. In the main pane, the "Columns" tab is highlighted in red. A table lists the columns of the "DEPARTMENTS" table:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
DEPARTMENT_ID	NUMBER(4,0)	No	(null)	1	1	Primary key column
DEPARTMENT_NAME	VARCHAR2(30 BYTE)	No	(null)	2		(null) A not null column th
MANAGER_ID	NUMBER(6,0)	Yes	(null)	3		(null) Manager_id of a dep
LOCATION_ID	NUMBER(4,0)	Yes	(null)	4		(null) Location id where a

At the bottom left is the ORACLE logo, and at the bottom right is the copyright notice: Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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: DESCRIBE is a SQL*Plus command supported by SQL Developer. It is abbreviated as DESC.

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)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

The data types are described in the following table:

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



Quiz

Identify the SELECT statements that execute successfully.

- a.

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

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

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

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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
- Describes the structure of a table



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

In the syntax:

SELECT	Is a list of one or more columns
*	Selects all columns
DISTINCT	Suppresses duplicates
column / expression	Selects the named column or the expression
alias	Gives different headings to the selected columns
FROM table	Specifies the table containing the columns

Practice 2: 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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

3

Restricting and Sorting Data



Copyright © 2017, Oracle and/or its affiliates. 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 © 2017, Oracle and/or its affiliates. All rights reserved.

When retrieving data from a 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 in the slide.

Lesson Agenda

- Limiting rows with:
 - The WHERE clause
 - The comparison operators 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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Limits Rows by Using a Selection

EMPLOYEES

	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	
4	103 Hunold	IT_PROG	60	
5	104 Ernst	IT_PROG	60	
6	107 Lorentz	IT_PROG	60	

“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	



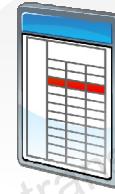
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Limits Rows That Are Selected

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

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

- The WHERE clause follows the FROM clause.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

In the syntax:

WHERE
logical expression

Restricts the query to rows that meet a condition
Is composed of column names, constants, and a comparison operator. It specifies a combination of one or more expressions and Boolean operators, and returns a value of TRUE, FALSE, or UNKNOWN.

The WHERE clause can compare literals and values in columns, arithmetic expressions, or functions.

It consists of three elements:

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

Using the WHERE Clause

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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Character Strings and Dates

- Character strings and date values are enclosed within single quotation marks (' ').
- Character values are case-sensitive and date values are format-sensitive.
- The default display format for date is DD-MON-RR.

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

LAST_NAME	JOB_ID	DEPARTMENT_ID
Whalen	AD_ASST	10

```
SELECT last_name
FROM employees
WHERE hire_date = '17-OCT-11' ;
```

LAST_NAME
Rajs



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You must enclose character strings and dates in the WHERE clause within single quotation marks (' '). Number constants, however, need not be enclosed within single quotation marks.

Remember that all character searches are case-sensitive. In the following example, observe that 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 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use comparison operators 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-05'  
... 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

Let us look at some examples:

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

```
SELECT *  
FROM   employees  
WHERE  last_name = 'Abel' ;
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-12	SA_REP	11000	0.3	149	80



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Range Conditions Using the BETWEEN Operator

You can 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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. Remember, 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 'Whalen' ;
```

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	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the IN operator to test for values among a specified set of values. The condition defined using the IN operator is also known as the *membership condition*.

The example in the slide displays employee IDs, last names, salaries, and manager's employee IDs for all the employees whose manager's employee ID 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 rows from the EMPLOYEES table, for any employee whose last name is included with the IN operator:

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

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

Pattern Matching Using the LIKE Operator

- You can use the `LIKE` operator to perform wildcard searches of valid string patterns.
- The search conditions can contain either literal characters or numbers:
 - `%` denotes zero or more characters.
 - `_` denotes one character.

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

FIRST_NAME
1 Shelley
2 Steven



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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
<code>%</code>	Represents any sequence of zero or more characters
<code>_</code>	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 2015 and December 2015:

```
SELECT last_name, hire_date
  FROM employees
 WHERE hire_date LIKE '%15';
```

Combining Wildcard Symbols

- You can combine the two wildcard symbols (% , _) 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.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

When you need to have an exact match for the actual % and _ characters, use the ESCAPE identifier. For example, to find the last name and job ID of all the employees whose job ID contains 'SA_ ', use the following statement:

```
SELECT last_name, job_id  
FROM employees  
WHERE job_id LIKE 'SA\_%' ESCAPE '\';
```

Using NULL Conditions

You can use the `IS NULL` operator to test for NULL values in a column.

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

LAST_NAME	MANAGER_ID
King	(null)



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

There are two types of NULL conditions:

- `IS NULL` tests for NULL values.
- `IS NOT NULL` tests for values that are not NULL.

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_name` and `manager_id` 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;
```

Defining Conditions Using Logical Operators

You can use the logical operators to filter the result set based on more than one condition or invert the result set.

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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	149 Zlotkey	SA_MAN	10500	
2	201 Hartstein	MK_MAN	13000	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 within 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	100 King	AD_PRES	24000	
2	101 Kochhar	AD_VP	17000	
3	102 De Haan	AD_VP	17000	
4	124 Mourgos	ST_MAN	5800	
5	149 Zlotkey	SA_MAN	10500	
6	174 Abel	SA_REP	11000	
7	201 Hartstein	MK_MAN	13000	
8	205 Higgins	AC_MGR	12008	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 both 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

NOT is used to negate a condition:

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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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	TRUE/FALSE

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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Rules of Precedence

Order	Operator
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 operator
8	AND logical operator
9	OR logical operator

You can use parentheses to override rules of precedence.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Rules of Precedence

```
SELECT last_name, department_id, salary
FROM employees
WHERE department_id = 60
OR department_id = 80
AND salary > 10000;
```

LAST_NAME	DEPARTMENT_ID	SALARY
1 Hunold	60	9000
2 Ernst	60	6000
3 Lorentz	60	4200
4 Zlotkey	80	10500
5 Abel	80	11000

1

```
SELECT last_name, department_id, salary
FROM employees
WHERE (department_id = 60
OR department_id = 80)
AND salary > 10000;
```

LAST_NAME	DEPARTMENT_ID	SALARY
1 Zlotkey	80	10500
2 Abel	80	11000

2



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

1. Precedence of the AND Operator: Example

In this example, there are two conditions:

- The first condition is that the department ID is 80 *and* the salary is greater than \$10,000.
- The second condition is that the department ID is 60.

Therefore, the SELECT statement reads as follows:

“Select the row if an employee’s department ID is 80 *and* earns more than \$10,000, *or* if the employee’s department ID is 60.”

2. Using Parentheses: Example

In this example, there are two conditions:

- The first condition is that the department ID is 80 *or* 60.
- The second condition is that the salary is greater than \$10,000.

Therefore, the SELECT statement reads as follows:

“Select the row if an employee’s department ID is 80 *or* 60, *and* if the employee earns more than \$10,000.”

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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the ORDER BY Clause

You can sort the retrieved rows with the ORDER BY clause:

- ASC: Ascending order, default
- DESC: Descending order

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

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1 De Haan	AD_VP	90	13-JAN-09
2 Kochhar	AD_VP	90	21-SEP-09
3 Higgins	AC_MGR	110	07-JUN-10
4 Gietz	AC_ACCOUNT	110	07-JUN-10
5 King	AD_PRES	90	17-JUN-11
6 Whalen	AD_ASST	10	17-SEP-11
7 Rajs	ST_CLERK	50	17-OCT-11

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The order of rows that are returned in a query result is undefined. The ORDER BY clause can be used to sort the rows. You can specify an expression, an alias, or a column position as the sort condition. You can specify multiple expressions in the ORDER BY clause. Oracle Database first sorts rows based on their values for the first expression. Rows with the same value for the first expression are then sorted based on their values for the second expression, and so on.

Syntax

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

In the syntax:

ORDER BY	Specifies the order in which the retrieved rows are displayed
ASC	Orders the rows in ascending order (this is the default order)
DESC	Orders the rows in descending order

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

Sorting

- Sorting in descending order:

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

1

- Sorting by column alias:

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

2



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The default sort order is ascending. Let us look at the other characteristics:

- 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 department_id.
2. You can also use a column alias in the ORDER BY clause. The slide example sorts the data by annual salary.

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 by using the column's numeric position:

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

3

- Sorting by multiple columns:

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

4



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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. You list the columns (or `SELECT` list column sequence numbers) in the `ORDER BY` clause, delimited by commas. The results are ordered by the first column, then the second, and so on for as many columns as the `ORDER BY` clause includes. If you want any results sorted in descending order, your `ORDER BY` clause must use the `DESC` keyword directly after the name or the number of the relevant column. 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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands

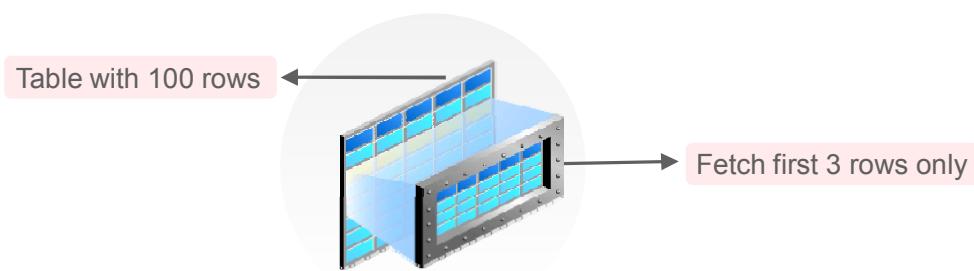


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL Row Limiting Clause

- You can use the `row_limiting_clause` to limit the rows that are returned by a query.
- You can use this clause to implement Top-N reporting.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Limiting the number of rows returned by a query can be valuable for reporting, analysis, data browsing, and other tasks.

With the new SQL `SELECT` syntax, you can limit the number of rows that are returned in the result set using the `row_limiting_clause`.

Queries that order data and then limit row output are widely used and are often referred to as Top-N queries. Top-N queries sort their result set and then return only the first n rows.

Note that there are certain limitations of the SQL `row_limiting_clause`:

- You cannot specify this clause with the `for_update_clause`.
- You cannot specify this clause in the subquery of a `DELETE` or `UPDATE` statement.
- If you specify this clause, then the `SELECT` list cannot contain the sequence pseudocolumns `CURRVAL` or `NEXTVAL`.

Using SQL Row Limiting Clause in a Query

You specify the `row_limiting_clause` in the SQL SELECT statement by placing it after the `ORDER BY` clause.

Syntax:

```
SELECT ...
  FROM ...
  [ WHERE ... ]
  [ ORDER BY ... ]
  [OFFSET offset { ROW | ROWS }]
  [FETCH { FIRST | NEXT } [{ row_count | percent PERCENT
  }] { ROW | ROWS }
  { ONLY | WITH TIES }]
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SQL Row Limiting Clause: Example

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
FETCH FIRST 5 ROWS ONLY;
```

Script Output X Query Result X	
SQL All Rows Fetched: 5	
EMPLOYEE_ID	FIRST_NAME
1	100 Steven
2	101 Neena
3	102 Lex
4	103 Alexander
5	104 Bruce

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

	EMPLOYEE_ID	FIRST_NAME
1	107 Diana	
2	124 Kevin	
3	141 Trenna	
4	142 Curtis	
5	143 Randall	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The first example in the slide returns the first five employees after sorting the rows in ascending order of the `employee_id`.

The second example in the slide returns the next set of five employees after sorting the rows in ascending order of the `employee_id`.

Note: If `employee_id` is assigned sequentially by the date when the employee joined the organization, these examples give us the top 5 employees and then employees 6-10, all in terms of seniority.

Lesson Agenda

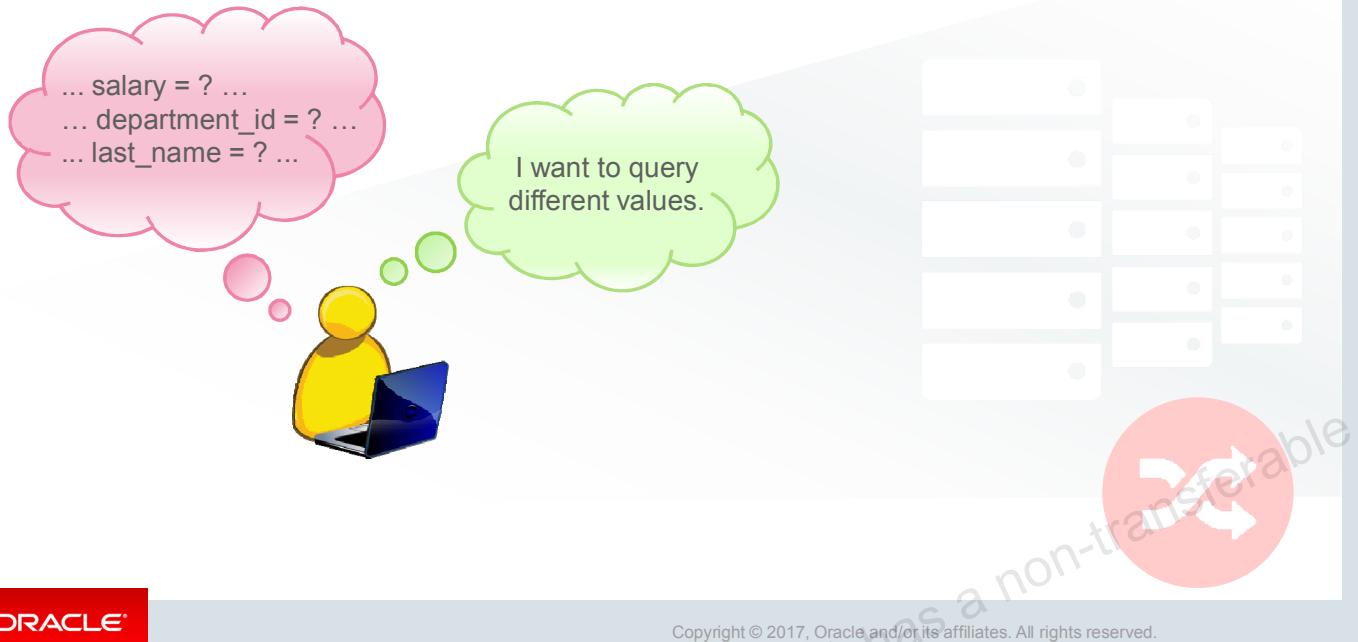
- Limiting rows with:
 - The WHERE clause
 - The comparison conditions using =, <=, BETWEEN, IN, LIKE, and NULL operators
 - Logical conditions using AND, OR, and NOT operators
- Rules of precedence for operators in an expression
- Sorting rows using the ORDER BY clause
- SQL row limiting clause in a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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Substitution Variables



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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. You can also use it for:

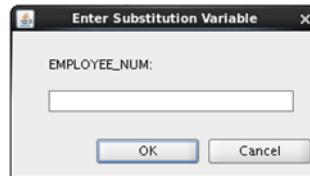
- Obtaining input values from a file rather than from a person
- Passing values from one SQL statement to another

Note: Both SQL Developer and SQL*Plus support substitution variables and the DEFINE/UNDEFINE commands.

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

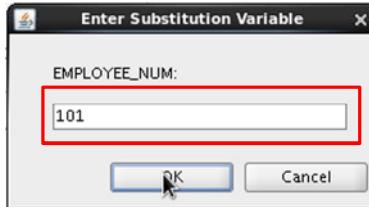
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When running a report, users often want to restrict the data that is returned dynamically. SQL*Plus and SQL Developer provide 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
&user_variable	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



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

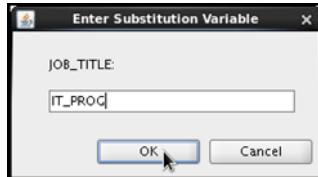
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:

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



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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

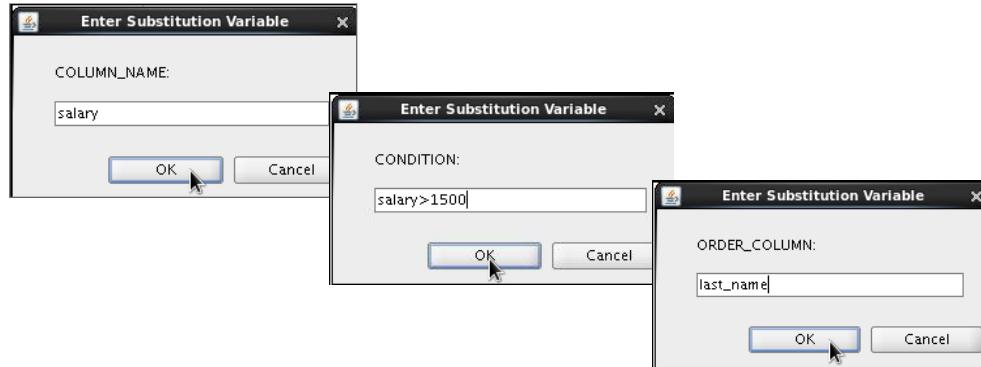
In a WHERE clause, you must enclose date and character values within single quotation marks. The same rule applies to the substitution variables.

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

```
SELECT employee_id, last_name, job_id,&column name  
FROM employees  
WHERE &condition  
ORDER BY &order column ;
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

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 ;
```

The screenshot shows the Oracle SQL Developer interface. At the top, there is a code editor window containing the SQL query with a substitution variable. Below it is a 'Enter Substitution Variable' dialog box where the user has entered a value for the variable. To the right of the dialog is a preview of the resulting data from the query.

Code in the editor:

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

Substitution Variable Dialog:

COLUMN_NAME:
department_id

Preview of the result:

	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 © 2017, Oracle and/or its affiliates. All rights reserved.

Using the Ampersand Substitution Variable in SQL*Plus

```
oracle@edcdr14p1:~$ SQL> SELECT employee_id, last_name, salary, department_id
  2  from employees
  3  where employee_id = &employee_num;
Enter value for employee num: 101
```

```
oracle@edcdr14p1:~$ SQL> SELECT employee_id, last_name, salary, department_id
  2  from employees
  3  where employee_id = &employee_num;
Enter value for employee num: 101
old  3: where employee_id = &employee_num
new  3: where employee_id = 101

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

SQL>
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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
- SQL row limiting clause in a query
- Substitution variables
- DEFINE and VERIFY commands



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the DEFINE Command

- Use the `DEFINE` command to create a variable and assign a value to it.
- 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
```

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the VERIFY Command

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

The screenshot shows the Oracle SQL Developer interface. At the top, a code editor window contains the following SQL statement with the `SET VERIFY ON` command highlighted:

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

Below the code editor is a modal dialog titled "Enter Substitution Variable". It has a single input field labeled "EMPLOYEE_NUM:" containing the value "200", with an "OK" button highlighted.

To the right of the dialog is the "Script Output" tab, which displays the results of the query execution. The output shows the original command with the substitution variable, followed by the modified command with the value substituted, and finally the resulting data from the database:

EMPLOYEE_ID	LAST_NAME	SALARY
200	Whalen	4400

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the VERIFY command to confirm the changes in the SQL statement. 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, on 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 four of the following are valid operators for the WHERE clause?

- a. >=
- b. IS NULL
- c. !=
- d. IS LIKE
- e. IN BETWEEN
- f. <>



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, c, f

Summary

In this lesson, you should have learned how to:

- 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 © 2017, Oracle and/or its affiliates. All rights reserved.

Practice 3: 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Using Single-Row Functions to Customize Output

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. 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 © 2017, Oracle and/or its affiliates. All rights reserved.

HR Application Scenario

How do I calculate the average salary of all employees working in China.



Zhen

HR Application			
Emp_ID	First Name	Salary	Location
101	Chang	10000	China
105	Xiu	15000	China
159	Tai	8000	China

Operation:

Average (Salary)

GO

HR Application

The average salary is \$10500.



- Accounts
- IT
- Sales
- Marketing

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Consider a scenario where Zhen, an HR manager in China, wants to calculate the average salary across various departments of all employees working in China. In order to generate such information, Zhen has to enter conditions such as country name (China) and get the list of employees working in China. Then he has to enter the operation to be performed on the values (in this case, `AVERAGE salary`). The HR application queries the database conditionally and then applies the mathematical operation to calculate the average salary. The results are returned to Zhen, along with a chart for analysis.

The operations performed on the values returned by a SQL query are called Functions. There are different types of functions, which are useful when you want to apply some kind of customization on the values returned by the query. In the following lessons, you will learn about the different types of functions.

Lesson Agenda

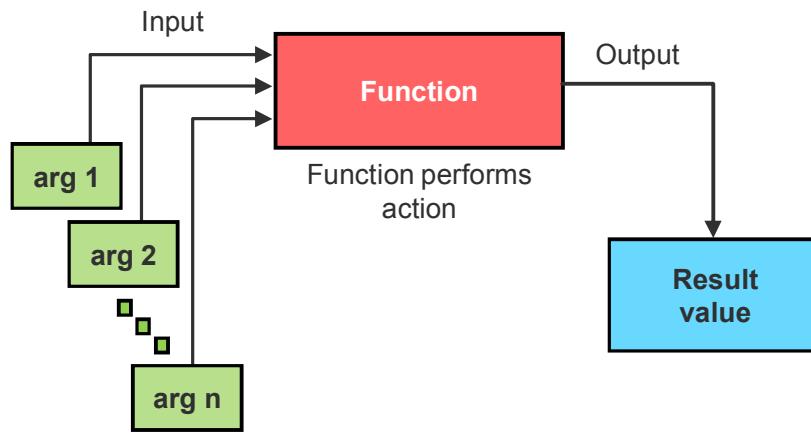
- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates
- Date functions



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

SQL Functions



ORACLE®

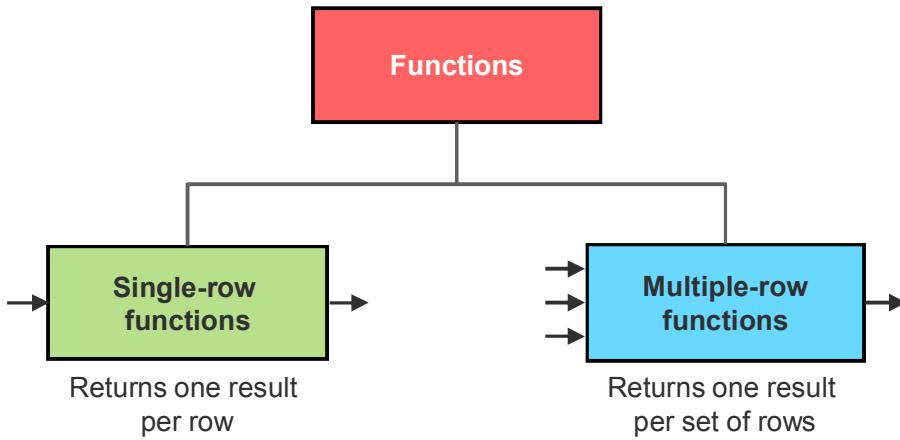
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Functions are a very powerful feature of SQL. You can use functions 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.

Two Types of SQL Functions



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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 “Functions” section in *Oracle Database SQL Language Reference* for 12c database.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

You can use single-row functions 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

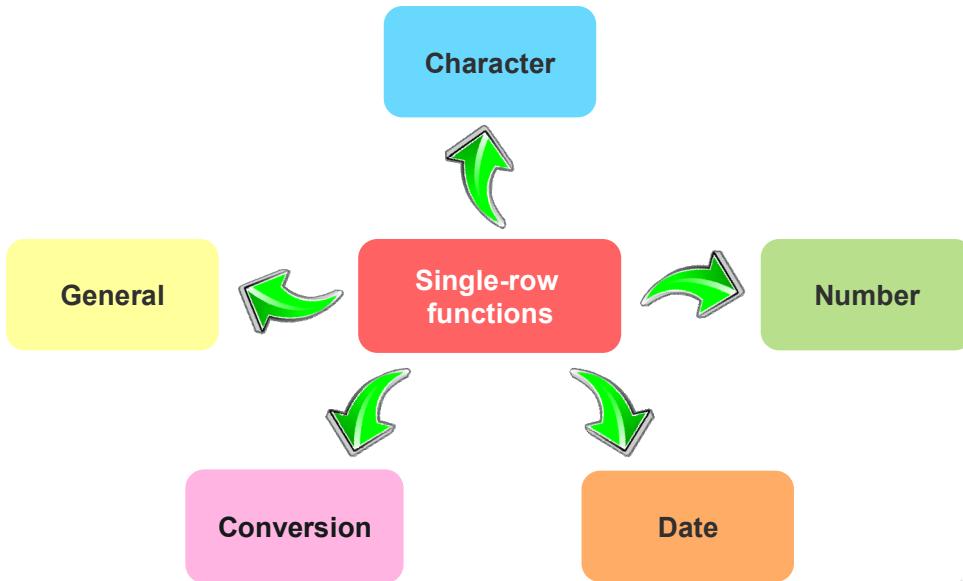
Features of single-row functions include the following:

- Act on each row that is returned in the query
- Return one result per row
- Possibly return a data value of a different type than the one that is referenced
- Possibly expect 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 © 2017, Oracle and/or its affiliates. All rights reserved.

In this lesson, you will learn about 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

You will learn about the following single-row functions in the lesson titled “Using Conversion Functions and Conditional Expressions”:

- **Conversion functions:** Convert a value from one data type to another
- **General functions:** These functions take any data type and can also handle NULLs.

Lesson Agenda

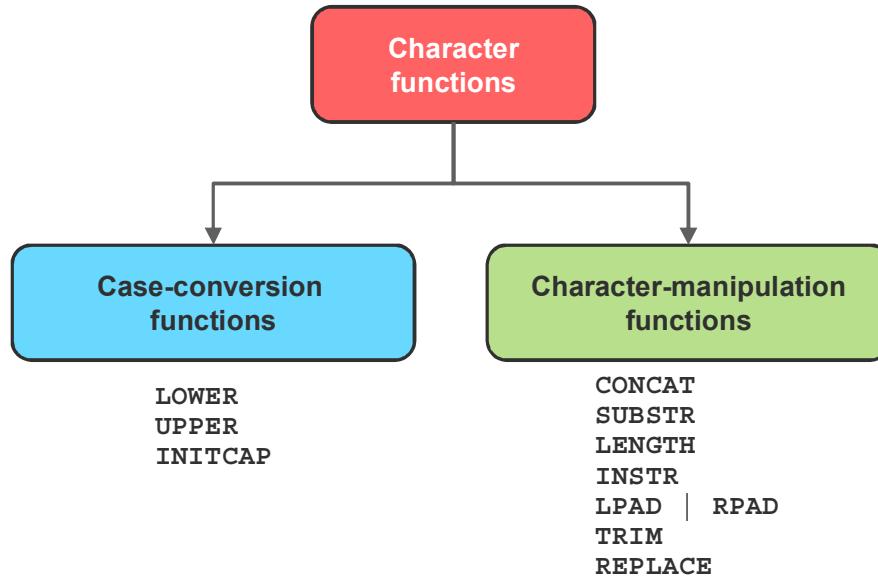
- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates
- Date functions



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Character Functions



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Function	Purpose
LENGTH(<i>column expression</i>)	Returns the number of characters in the expression
INSTR(<i>column 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 expression</i> , <i>n</i> , ' <i>string</i> ') RPAD(<i>column 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 trailing 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.
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

Case-Conversion Functions

You can use these functions to convert the case of character strings:

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

As shown in the slide, 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

For example:

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

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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 converts 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';
```

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.

Character-Manipulation Functions

You can use these functions to manipulate character strings:

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD (24000,10,'*')	*****24000
RPAD (24000, 10, '*)	24000****



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using Character-Manipulation Functions

```
SELECT last_name, CONCAT('Job category is ', job_id)
  "Job" FROM employees
 WHERE SUBSTR(job_id, 4) = 'REP';
```

1

LAST_NAME	JOB
Abel	Job category is SA_REP
Fay	Job category is MK_REP
Grant	Job category is SA_REP
Taylor	Job category is SA_REP

```
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';
```

2

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The first example in the slide displays employee last names and job IDs for all employees who have the string, REP, contained in the job ID, starting at the fourth position of the job ID.

The second SQL statement in the slide displays data such as employee ID, concatenated first name and last name, length of the last name, and the position of the first occurrence of the letter 'a' in the last name for those employees whose last names end with the letter "n."

Lesson Agenda

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

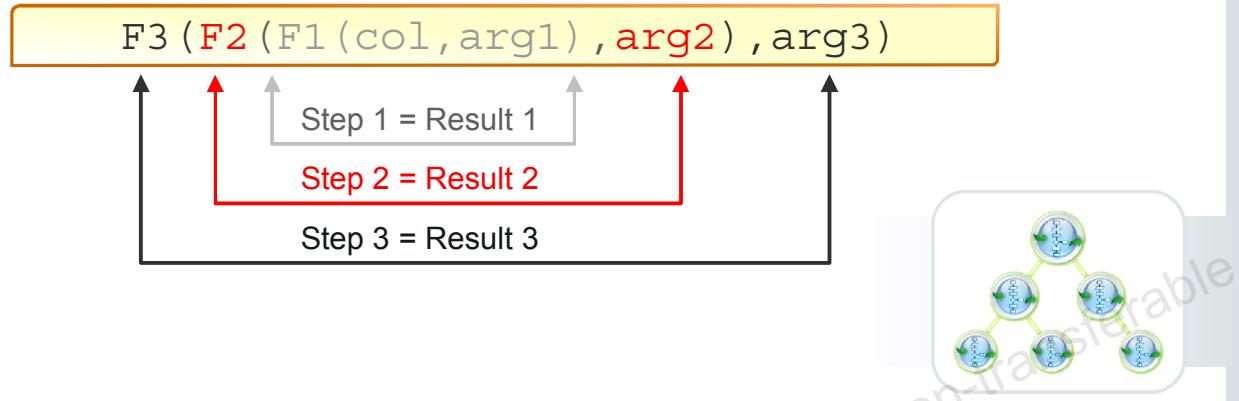


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Result3 = UPPER(Result2)

Result3 is displayed. The entire expression becomes the column heading because no column alias was given.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Numeric Functions

- **ROUND:** Rounds value to a specified decimal
- **TRUNC:** Truncates value to a specified decimal
- **CEIL:** Returns the smallest whole number greater than or equal to a specified number
- **FLOOR:** Returns the largest whole number equal to or less than a specified number
- **MOD:** Returns remainder of division

Function	Result
ROUND (45.926, 2)	45.93
TRUNC (45.926, 2)	45.92
CEIL (2.83)	3
FLOOR (2.83)	2
MOD (1600, 300)	100



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

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

For more information, see the “Numeric Functions” section in *Oracle Database SQL Language Reference* for 12c database.

Using the ROUND Function

```

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

	ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
1	45.92	46	50
2			
3			



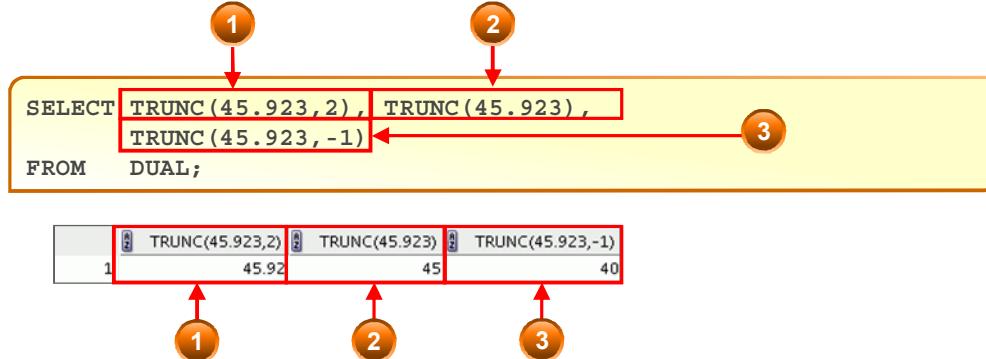
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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).

Recall 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Using the MOD Function

Display the employee records where the `employee_id` is an even number:

```
SELECT employee_id AS "Even Numbers", last_name  
FROM employees  
WHERE MOD(employee_id, 2) = 0;
```

#	Even Numbers	LAST_NAME
1	174 Abel	
2	142 Davies	
3	102 De Haan	
4	104 Ernst	
5	202 Fay	
6	206 Gietz	
7	178 Grant	
8	100 King	
9	124 Mourgos	
10	176 Taylor	
11	144 Vargas	
12	200 Whalen	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

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-2013';
```

LAST_NAME	HIRE_DATE
King	17-JUN-11
Kochhar	21-SEP-09
De Haan	13-JAN-09
...	

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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-11 is displayed as day, month, and year, there is also *time* and *century* information associated with the date. The complete date might be June 17, 2011, 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.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 preceding table.

This data is stored internally as follows:

CENTURY SECOND	YEAR	MONTH	DAY	HOUR	MINUTE
19 43	03	06	17	17	10

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). You will learn how to use the SYSDATE function in the next slide.

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

Use the SYSDATE function to get:

- Date
- Time

```
SELECT sysdate  
FROM   dual;
```

 SYSDATE
1 23-JUN-16



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

SYSDATE is a date function that returns the system date. You can use SYSDATE just as you would use any other column name. For example, you can display the system 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 such a case, to get the local time, you can use the CURRENT_DATE function that returns the current date in the session time zone.

Using the CURRENT_DATE and CURRENT_TIMESTAMP Functions

- CURRENT_DATE returns the current date from the user session.

```
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;
```

SESSIONTIMEZONE	CURRENT_DATE
1 Etc/Universal	23-JUN-16

- CURRENT_TIMESTAMP returns the current date and time from the user session.

```
SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL;
```

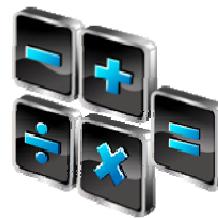
SESSIONTIMEZONE	CURRENT_TIMESTAMP
1 Etc/Universal	23-JUN-16 01.26.18.154099000 AM ETC/UNIVERSAL



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Arithmetic with Dates

- Add to or subtract a number 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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	478.871917989417989417989417989417989418
2	Kochhar	360.729060846560846560846560846560846561
3	De Haan	605.300489417989417989417989417989417989



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Date-Manipulation Functions

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 preceding 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-16' , '11-JAN-15')	19.6774194
ADD_MONTHS ('31-JAN-16' , 1)	'29-FEB-16'
NEXT_DAY ('01-JUN-16' , 'FRIDAY')	'03-JUN-16'
LAST_DAY ('01-APR-16')	'30-APR-16'



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using ROUND and TRUNC Functions with Dates

Assumption: The date when the below functions were run was **08-JUL-16**.

Function	Result
ROUND(SYSDATE, 'MONTH')	01-JUL-16
ROUND(SYSDATE, 'YEAR')	01-JAN-17
TRUNC(SYSDATE, 'MONTH')	01-JUL-16
TRUNC(SYSDATE, 'YEAR')	01-JAN-16



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 2010. 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 '%10';
```



Quiz

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

- a. Manipulate data items
- b. Accept arguments and return one value per argument
- c. Act on each row that is returned
- d. Return one result per set of rows
- e. Never modify the data type
- f. Can be nested
- g. Accept arguments that can be a column or an expression



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Practice 4: Overview

This practice covers the following topics:

- Writing a query that displays the SYSDATE
- 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 © 2017, Oracle and/or its affiliates. All rights reserved.

Using Conversion Functions and Conditional Expressions

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. 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 © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

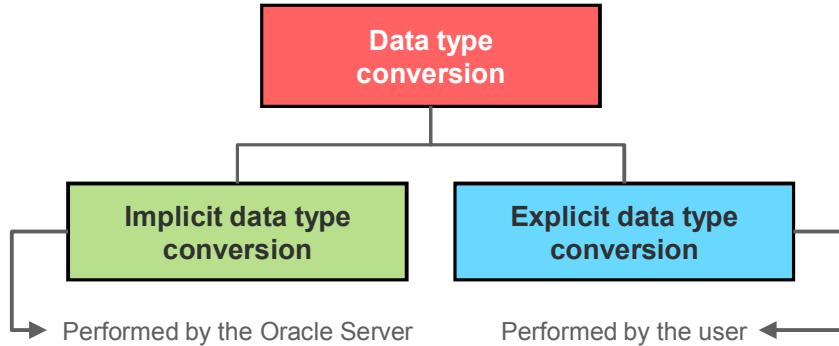
- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- General functions:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- Conditional expressions:
 - CASE
 - Searched CASE
 - DECODE



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Conversion Functions



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

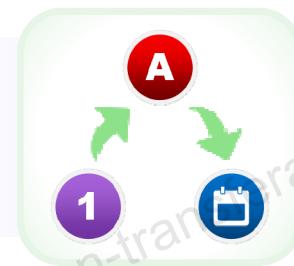
Note: Although implicit data type conversion is available, it is recommended that you do the explicit data type conversion to ensure the reliability of your SQL statements.

In addition to Oracle data types, you can define the columns of tables in an Oracle Database 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.

Implicit Data Type Conversion of Strings

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

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Implicit Data Type Conversion to Strings

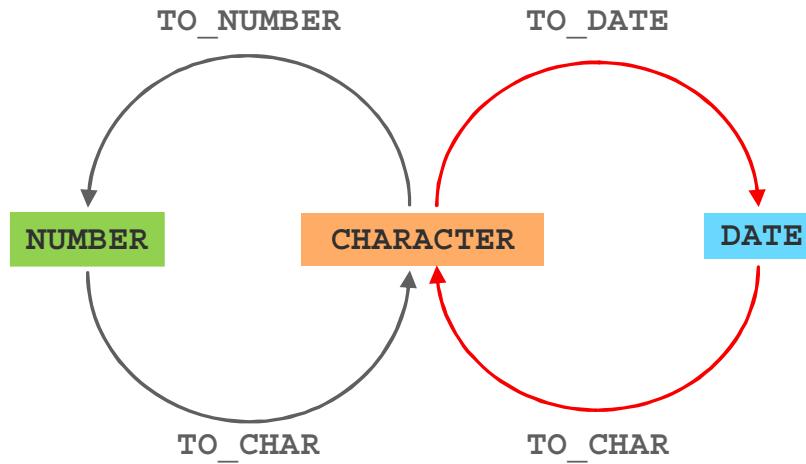
For expression evaluation, the Oracle server can automatically convert the following:

From	To
NUMBER	VARCHAR2 or CHAR
DATE	VARCHAR2 or CHAR



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Explicit Data Type Conversion



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Let us look at the three functions SQL provides to convert a value from one data type to another:

Function	Purpose
<code>TO_CHAR (number/date [, fmt [, nlsparams]])</code>	<p>Converts a number or date value to a VARCHAR2 character string with the format model <i>fmt</i></p> <p>Number conversion: The <i>nlsparams</i> argument 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>

Function	Purpose
TO_NUMBER (<i>char</i> [, <i>fmt</i> [, <i>nlsparams</i>]])	Converts a character string containing digits to a number in the format specified by the optional format model <i>fmt</i> . The <i>nlsparams</i> parameter has the same purpose in this function as in the TO_CHAR function for number conversion.
TO_DATE (<i>char</i> [, <i>fmt</i> [, <i>nlsparams</i>]])	Converts a character string representing a date to a date value according to <i>fmt</i> that is specified. If <i>fmt</i> is omitted, the format is DD-MON-YY. The <i>nlsparams</i> parameter has the same purpose in this function as in the TO_CHAR function for date conversion.

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

For more information, see the “Conversion Functions” section in *Oracle Database SQL Language Reference for 12c database*.

Lesson Agenda

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the TO_CHAR Function with Dates

Example:

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

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

EMPLOYEE_ID	MONTH_HIRED
1	205 06/10



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can convert a datetime data type to a value of VARCHAR2 data type using TO_CHAR 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-2000' 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.

Here are some of the guidelines to follow while using TO_CHAR:

- Enclose the format model within single quotation marks. The format model is case-sensitive.
- Ensure that you separate the date value from the format model with a comma. The format model can include any valid date format element.
- The names of days and months in the output are automatically padded with blanks.
- Use the fill mode *fm* element to remove padded blanks or to suppress leading zeros.

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Element	Description
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYY	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 help you format the time portion of the date:

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

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

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

- Number suffixes help in spelling out numbers:

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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	12 hour format
HH24	24 hour format
MI	Minute (0–59)
SS	Second (0–59)
SSSS	Seconds past midnight (0–86399)

Other Formats

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

Specifying Suffixes to Influence Number Display

Element	Description
TH	Ordinal number (for example, DDTH for 4TH)
SP	Spelled-out number (for example, 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 King	17 June 2011
2 Kochhar	21 September 2009
3 De Haan	13 January 2009
4 Hunold	3 January 2014
5 Ernst	21 May 2015
6 Lorentz	7 February 2015
7 Mourgos	16 November 2015
8 Rajs	17 October 2011

...



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The SQL statement in the slide displays the last names and hire dates for all the employees. Observe that the hire date appears as 17 June 2011.

Example

Modify the example in the slide to display the dates in a format that appears as “Seventeenth of June 2011 12:00:00 AM.”

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

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

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

`TO_CHAR(number[, 'format_model'])`

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Number Format Elements

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

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
\$	Floating dollar sign	\$999999	\$1234
L	Floating local currency symbol	L999999	FF1234
D	Returns the decimal character in the specified position. The default is a period (.).	9999D99	1234.00
.	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.	9G999	1,234
,	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 n times (n = 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

Let us look at an example:

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

	SALARY
1	\$6,000.00



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

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'])
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Example

Display the name and hire date for all employees who started on May 24, 2015. 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, 2015', 'fxMonth DD, YYYY');
```

The resulting error output looks like this:

```
ORA-01858: a non-numeric character was found where a numeric was expected  
01858.00000 - "a non-numeric character was found where a numeric was expected"  
*Cause: The input data to be converted using a date format model was  
incorrect. The input data did not contain a number where a number was  
required by the format model.  
*Action: Fix the input data or the date format model to make sure the  
elements match in number and type. Then retry the operation.
```

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, 2015', 'fxMonth DD, YYYY');
```

Using TO_CHAR and TO_DATE Functions with the RR Date Format

To find employees hired before 2010, use the RR date format, which produces the correct result if the command is run now or before the year 2049:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01 Jan, 10', 'DD Mon,RR');
```

LAST_NAME	TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
Kochhar	21-Sep-2009
De Haan	13-Jan-2009



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

To find employees who were hired before 2010, 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 2000 to 2049. Alternatively, the following command also results in the same rows being selected because the YY format interprets the year portion of the date in the current century (2010).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM employees
WHERE hire_date < '01-Jan-10';
```

Notice that the same rows are retrieved from the preceding query.

The general tip is to use the YYYY format instead of RR format to avoid confusion.

Lesson Agenda

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

General Functions

The following functions pertain to using nulls and can be used with any data type:

A

NVL (expr1, expr2)

B

NULLIF (expr1, expr2)

C

NVL2 (expr1, expr2, expr3)

D

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

E

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

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

Note: For more information about the hundreds of functions available, see the “Functions” section in *Oracle Database SQL Language Reference* for 12c database.

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.
- Examples:
 - `NVL(commission_pct, 0)`
 - `NVL(hire_date, '01-JAN-97')`
 - `NVL(job_id, 'No Job Yet')`

`NVL (expr1, expr2)`



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 with 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	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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;
```

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

NVL2 (expr1, expr2, expr3)

```
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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 be of any data type, but *expr2* and *expr3* should be of the same data type.

Using the NULLIF Function

NULLIF (expr1, expr2)

```
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)
...					



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 with 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.

Using the COALESCE Function

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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

Syntax

COALESCE (expr1, expr2, ... exprn)

In the syntax:

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

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

Using the COALESCE Function

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

#	LAST_NAME	SALARY	COMMISSION_PCT	NewSalary
1	King	24000	(null)	26000
2	Kochhar	17000	(null)	19000
3	De Haan	17000	(null)	19000
4	Hunold	9000	(null)	11000
5	Ernst	6000	(null)	8000
6	Lorentz	4200	(null)	6200
7	Mourgos	5800	(null)	7800
8	Rajs	3500	(null)	5500
9	Davies	3100	(null)	5100
10	Matos	2600	(null)	4600
11	Vargas	2500	(null)	4500
12	Zlotkey	10500	0.2	12600
13	Abel	11000	0.3	14300
14	Taylor	8600	0.2	10320
15	Grant	7000	0.15	8050
16	Whalen	4400	(null)	6400
17	Hartstein	13000	(null)	15000
18	Fay	6000	(null)	8000
19	Higgins	12008	(null)	14008
20	Gietz	8300	(null)	10300



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example shown in the slide, 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.

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.

Lesson Agenda

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

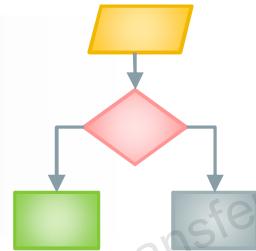


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Conditional Expressions

- Help provide the use of IF-THEN-ELSE logic within a SQL statement
- You can use the following methods:
 - CASE expression
 - Searched CASE expression
 - DECODE function



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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, NVARCHAR2 , NUMBER, BINARY_FLOAT, or BINARY_DOUBLE or must all have a numeric data type. All of the return values (return_expr) must be of the same data type.

Using the CASE Expression

```
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
King	AD_PRES	24000	24000
...			
4 Hunold	IT_PROG	9000	9900
5 Ernst	IT_PROG	6000	6600
6 Lorentz	IT_PROG	4200	4620
7 Mourgos	ST_MAN	5800	5800
8 Rajs	ST_CLERK	3500	4025
9 Davies	ST_CLERK	3100	3565
10 Matos	ST_CLERK	2600	2990
11 Vargas	ST_CLERK	2500	2875
...			
13 Abel	SA REP	11000	13200
14 Taylor	SA REP	8600	10320
15 Grant	SA REP	7000	8400



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Searched CASE Expression

```
CASE
    WHEN condition1 THEN use_expression1
    WHEN condition2 THEN use_expression2
    WHEN condition3 THEN use_expression3
    ELSE default_use_expression
END
```

```
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;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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. The searched CASE expression evaluates the conditions independently under each of the WHEN options.

The difference between the CASE expression and the searched CASE expression is that in a searched CASE expression, you specify a condition or predicate instead of a comparison_expression after the WHEN keyword.

For both simple and searched CASE expressions, all of the return_exprs must either have the same data type CHAR, VARCHAR2, NCHAR, NVARCHAR2, NUMBER, BINARY_FLOAT, or BINARY_DOUBLE or must all have a numeric data type.

The code in the slide is an example of the searched CASE expression.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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
...				
4 Hunold	IT_PROG	9000	9900	
5 Ernst	IT_PROG	6000	6600	
6 Lorentz	IT_PROG	4200	4620	
7 Mourgos	ST_MAN	5800	5800	
8 Raji	ST_CLERK	3500	4025	
9 Davies	ST_CLERK	3100	3565	
10 Matos	ST_CLERK	2600	2990	
11 Vargas	ST_CLERK	2500	2875	
12 Zlotkey	SA_MAN	10500	10500	
...				
13 Abel	SA REP	11000	13200	
14 Taylor	SA REP	8600	10320	
15 Grant	SA REP	7000	8400	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the DECODE Function

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

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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:

Monthly Salary Range	Tax Rate
\$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%

Quiz



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

- a. True
- b. False



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: b

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 © 2017, Oracle and/or its affiliates. All rights reserved.

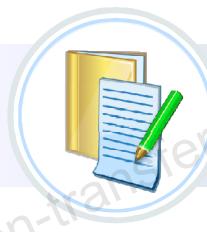
Remember the following:

- 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, searched CASE, or the DECODE function.

Practice 5: 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 `CASE`, `searched CASE`, and `DECODE`



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

This practice provides a variety of exercises using the `TO_CHAR` and `TO_DATE` functions, and conditional expressions such as `CASE`, `searched CASE`, and `DECODE`.

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

Reporting Aggregated Data Using the Group Functions

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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

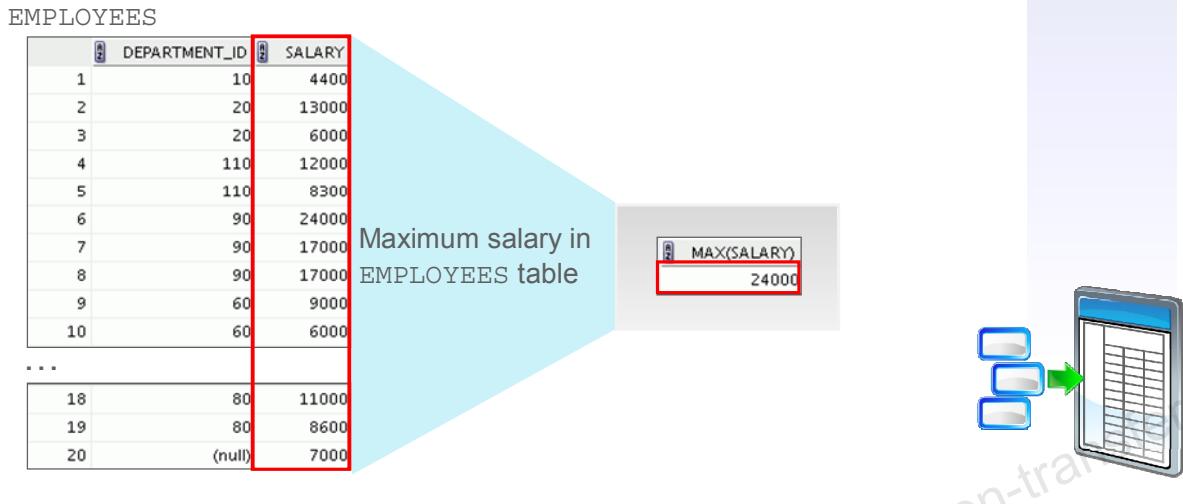


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Group Functions

Group functions operate on sets of rows to give one result per group.



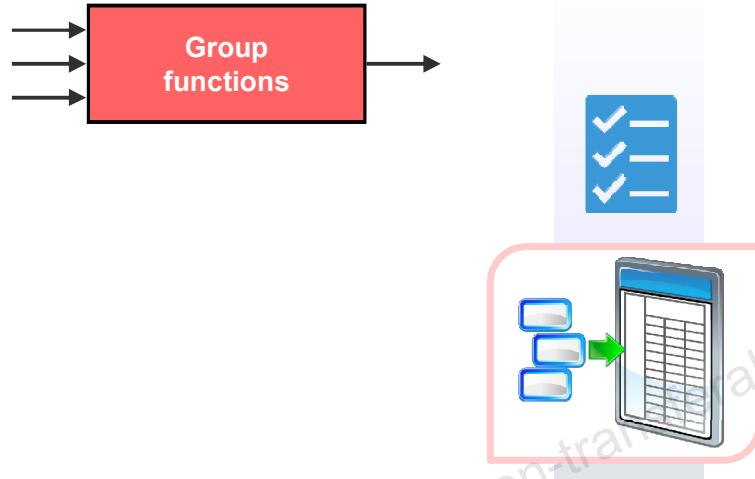
ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 groups of rows in the table.

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- VARIANCE



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

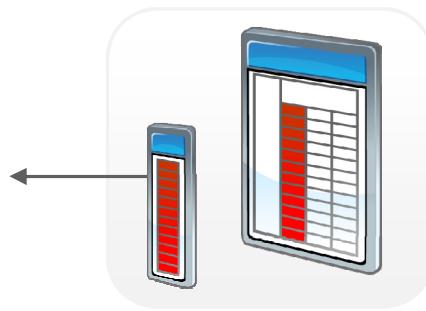
Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	Description
AVG ([DISTINCT ALL] <i>n</i>)	Average value of <i>n</i> , ignoring null values
COUNT	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 ALL] <i>expr</i>)	Maximum value of <i>expr</i> , ignoring null values
MIN ([DISTINCT ALL] <i>expr</i>)	Minimum value of <i>expr</i> , ignoring null values
STDDEV ([DISTINCT ALL] <i>n</i>)	Standard deviation of <i>n</i> , ignoring null values
SUM ([DISTINCT ALL] <i>n</i>)	Sum values of <i>n</i> , ignoring null values
LISTAGG	Orders data within each group specified in the ORDER BY clause and then concatenates the values of the measure column
VARIANCE ([DISTINCT ALL] <i>n</i>)	Variance of <i>n</i> , ignoring null values

Group Functions: Syntax

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

Group all rows in a column



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

Syntax:

`group_function([DISTINCT | ALL] expr)`

Let us look at a few 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 the AVG and SUM functions 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 13-JAN-09	29-JAN-16



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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;
```

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:

```
SELECT COUNT(*)
FROM employees
WHERE department_id = 50;
```

	COUNT(*)
1	5

- COUNT (expr) returns the number of rows with non-null values for expr:

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 50;
```

	COUNT(COMMISSION_PCT)
1	0



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the COUNT function in the following 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

- The first example in the slide displays the number of employees in department 50.
- The second example in the slide displays the number of employees in department 50 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Group Functions and Null Values

- Group functions ignore null values in the column:

```
SELECT AVG(commission_pct)
FROM employees;
```

1

	AVG(COMMISSION_PCT)
1	0.2125

- The NVL function forces group functions to include null values:

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

2

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

Creating Groups of Data

EMPLOYEES

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

Average salary in the EMPLOYEES table for each department

DEPARTMENT_ID	AVG(SALARY)
1	(null)
2	20
3	90
4	110
5	50
6	80
7	10
8	60



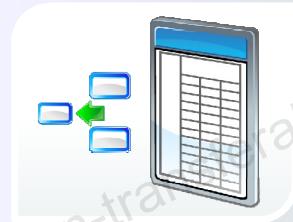
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Until this point in the discussion, you have observed that 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 the 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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

Let us look at some guidelines for using the GROUP BY clause:

- If you include a group function in a SELECT clause, you cannot select an individual column 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 can substitute *column* with an expression in the SELECT statement.
- 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	90	19333.33333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.33333333333333333333333333333333
7	60	6400
8	10	4400



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When using the GROUP BY clause, ensure 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 ;
```

	AVG(SALARY)
1	7000
2	19333.3333333333333333333333333333333333
3	9500
4	10154
5	3500
6	10033.3333333333333333333333333333333333
7	6400
8	4400



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You do not have to include the GROUP BY column 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) ;
```

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	12008
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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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	12008



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 columns 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;
```

1

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;
```

2

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

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

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- 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;
```

```
ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9
```

Cannot use the
WHERE clause to
restrict groups



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You cannot use the WHERE clause 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;
```

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 maximum salary for each department by grouping by department number.
2. Restrict the groups to the 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];
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the HAVING Clause

```
SELECT      department_id, MAX(salary)
FROM        employees
GROUP BY    department_id
HAVING     MAX(salary)> 10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the HAVING Clause

```
SELECT      job_id, SUM(salary) PAYROLL
FROM        employees
WHERE       job_id NOT LIKE '%REP%'
GROUP BY   job_id
HAVING     SUM(salary) > 13000
ORDER BY   SUM(salary);
```

JOB_ID	PAYROLL
1 IT_PROG	19200
2 AD_PRES	24000
3 AD_VP	34000



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

Nesting Group Functions

Display the maximum average salary:

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

	MAX(AVG(SALARY))
1	19333.3333333333333333333333333333333333333

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 the GROUP BY clause is mandatory when nesting group functions.

Quiz



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

- a. You cannot use a column alias in the GROUP BY clause.
- b. The GROUP BY column must be in the SELECT clause.
- c. By using a WHERE clause, you can exclude rows before dividing them into groups.
- d. The GROUP BY clause groups rows and ensures the order of the result set.
- e. If you include a group function in a SELECT clause, you must include a GROUP BY clause.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

Summary

In this lesson, you should have learned how to:

- Use the group functions COUNT, MAX, MIN, SUM, AVG, LISTAGG, STDDEV, and VARIANCE
- 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 © 2017, Oracle and/or its affiliates. All rights reserved.

There are several group functions available in SQL, such as AVG, COUNT, MAX, MIN, SUM, LISTAGG, 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. You can have either the GROUP BY clause or the HAVING clause first, as long as they follow the WHERE clause. 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 group functions, see *Oracle Database SQL Language Reference* for 12c database.

Practice 6: Overview

This practice covers the following topics:

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

Displaying Data from Multiple Tables Using Joins

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. 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 by 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 © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cartesian product
 - Cross join

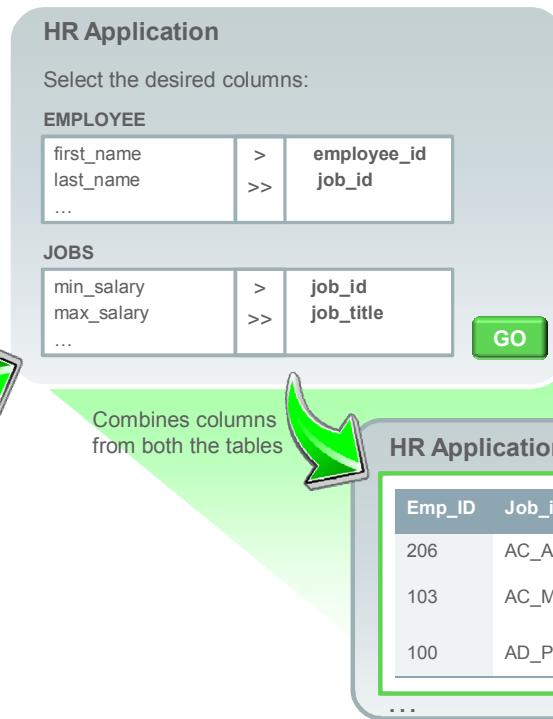


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Why Join?

I want the information of all employees and their jobs. But they are stored in different tables. How to I combine them into a single report?



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

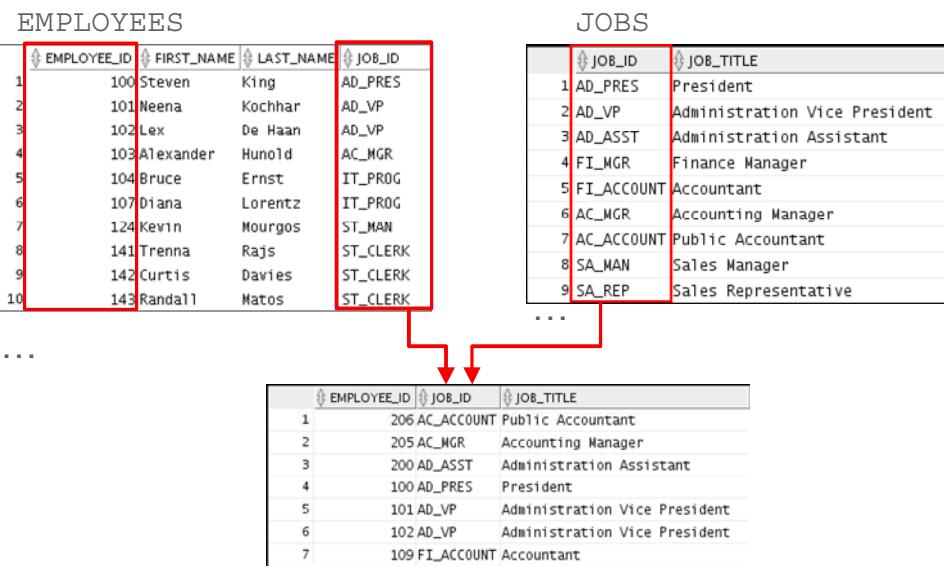
Jody, an HR Manager located in Mexico, wants a report of all employees working in the organization, their respective job IDs, and job titles. You cannot get this report by querying a single table because the employee information is in one table and the job information in another table.

What is the solution for Jody's query?

You will need to write a SQL statement to fetch the required information from two different tables. This SQL statement will fetch the employee IDs from the EMPLOYEE table and Job Title from the JOBS table by using JOB_ID as the common column. The result of the SQL Query is a single report consisting of the following columns: Employee ID, Job ID, and Job title.

The following slides explain in detail about Joins and how to use them.

Obtaining Data from Multiple Tables


ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.
- Job IDs exist in both the EMPLOYEES and JOBS tables.
- Job titles exist in the JOBS table.

To produce the report, you need to link the EMPLOYEES and JOBS tables, and access data from both of them.

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Natural join with the NATURAL JOIN clause
- Join with the USING clause
- Join with the ON clause
- OUTER joins:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cross joins



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT table1.column, table2.column
FROM table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name =
table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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* for 12c database.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cartesian product
 - Cross join



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Creating Natural Joins

- The NATURAL JOIN clause is based on all the columns that have the same name in two tables.
- 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.

```
SELECT * FROM table1 NATURAL JOIN table2;
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Retrieving Records with Natural Joins

```
SELECT employee_id, first_name, job_id, job_title
from employees NATURAL JOIN jobs;
```

EMPLOYEE_ID	FIRST_NAME	JOB_ID	JOB_TITLE
1	206 William	AC_ACCOUNT	Public Accountant
2	205 Shelley	AC_MGR	Accounting Manager
3	200 Jennifer	AD_ASST	Administration Assistant
4	100 Steven	AD_PRES	President
5	102 Lex	AD_VP	Administration Vice President
6	101 Neena	AD_VP	Administration Vice President
7	103 Alexander	IT_PROG	Programmer
8	104 Bruce	IT_PROG	Programmer
9	107 Diana	IT_PROG	Programmer
10	201 Michael	MK_MAN	Marketing Manager
11	202 Pat	MK_REP	Marketing Representative
12	149 Eleni	SA_MAN	Sales Manager
13	174 Ellen	SA REP	Sales Representative
14	178 Kimberly	SA REP	Sales Representative
15	176 Jonathon	SA REP	Sales Representative
16	143 Randall	ST_CLERK	Stock Clerk
17	142 Curtis	ST_CLERK	Stock Clerk
18	141 Trenna	ST_CLERK	Stock Clerk
19	144 Peter	ST_CLERK	Stock Clerk
20	124 Kevin	ST_MAN	Stock Manager



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, observe that the JOBS table is joined to the EMPLOYEES table by the JOB_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

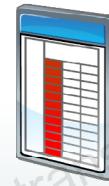
Additional restrictions on a natural join are implemented by using a WHERE clause. The following example limits the rows of output to those with a DEPARTMENT_ID equal to 20 or 50:

```
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

When should you use the USING clause?

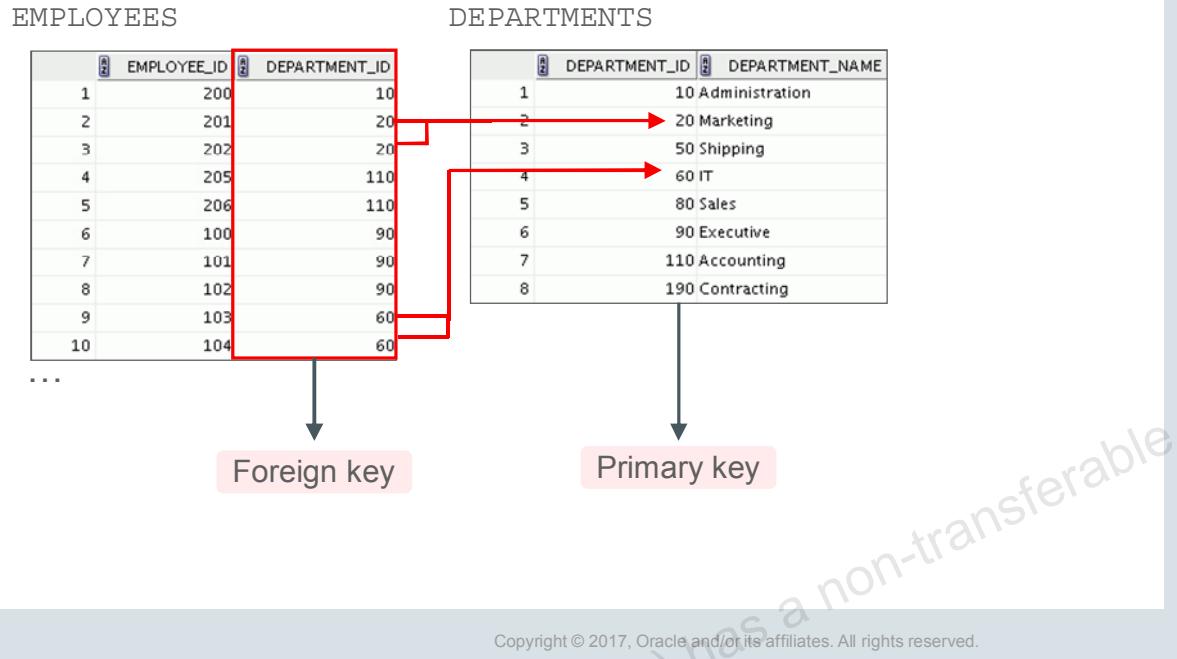
- If several columns have the same names but the data types do not match, use the USING clause to specify the columns for the equijoin.
- Use the USING clause to match only one column when more than one column matches.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Joining Column Names


ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

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 © 2017, Oracle and/or its affiliates. All rights reserved.

Qualifying Ambiguous Column Names

- Use table prefixes to:
 - Qualify column names that are in multiple tables
 - Increase the speed of parsing of a statement
- 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 © 2017, Oracle and/or its affiliates. All rights reserved.

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without 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. Basically, using the table prefix increases the speed of parsing of the statement, 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, use 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.

Using Table Aliases with the USING Clause

- Do not qualify a column that is used in the NATURAL join or a join with a 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;
```

ORA-25154: column part of USING clause cannot have qualifier
 25154. 00000 - "column part of USING clause cannot have qualifier"
 *Cause: Columns that are used for a named-join (either a NATURAL join
 or a join with a USING clause) cannot have an explicit qualifier.
 *Action: Remove the qualifier.
 Error at Line: 4 Column: 6



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 the columns to join.
- Use the `ON` clause to separate the join condition from other search conditions.
- The `ON` clause makes code easy to understand.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Retrieving Records with the ON Clause

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400
...					



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 parentheses 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

```
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

...



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 ;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

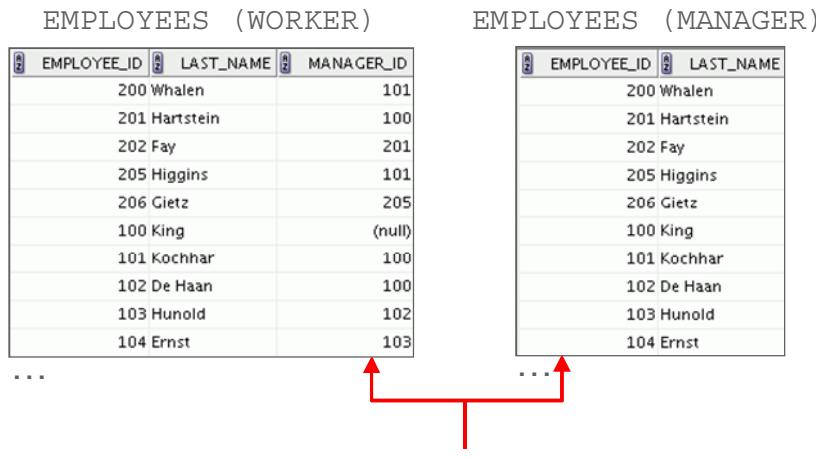
- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cartesian product
 - Cross join



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Joining a Table to Itself



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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 Ernst's manager, you need to:

- Find Ernst in the EMPLOYEES table by looking at the LAST_NAME column
- Find the manager number for Ernst by looking at the MANAGER_ID column. Ernst'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 Ernst's manager.

In this process, you look in the table twice. The first time you look in the table to find Ernst 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
8 Kochhar	King

...

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cartesian product
 - Cross join



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Nonequi joins

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES

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

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 based on his salary.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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;
```

#	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C
...			



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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_GRADES table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.

Note: Other conditions (such as `<=` and `>=`) can be used, but `BETWEEN` is the simplest. Remember to specify the low value first and the high value last when using 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 example in the slide for performance reasons, not because of possible ambiguity.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cartesian product
 - Cross join



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

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

There are no employees in department 190.

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

Therefore, the above two records do not appear in the equijoin result.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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

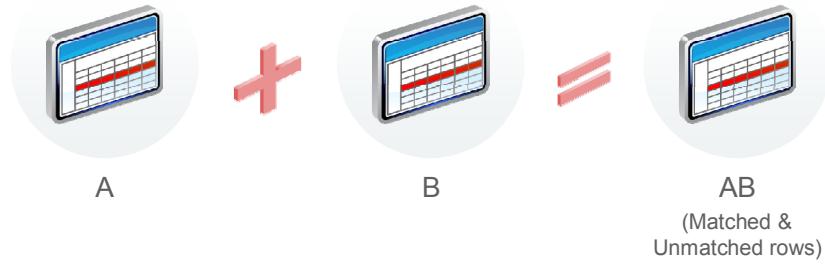
In the example in the slide, a simple equijoin condition is used on the EMPLOYEES and DEPARTMENTS tables to return the result on the right. The result set does not contain the following:

- 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 © 2017, Oracle and/or its affiliates. All rights reserved.

Joining tables with the **NATURAL JOIN**, **USING**, or **ON** clause 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)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT
...			

15	Grant	(null)(null)
16	Whalen	10 Administration
17	Hartstein	20 Marketing
18	Fay	20 Marketing
19	Higgins	110 Accounting
20	Gietz	110 Accounting
21	(null)	190 Contracting



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Types of JOINS and their syntax
- Natural join
- Join with the USING clause
- Join with the ON clause
- Self-join
- Nonequijoins
- OUTER join:
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- Cartesian product
 - Cross join



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cartesian Products

A Cartesian product:

- Is a join of every row of one table to every row of another table
- Generates a large number of rows and the result is rarely useful



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Creating Cross Joins

- A CROSS JOIN is a JOIN operation that produces a Cartesian product of two tables.
- To create a Cartesian product, specify CROSS JOIN in your SELECT statement.

```
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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



Quiz

If you join a table to itself, what kind of join are you using?

- a. Nonequijoin
- b. Left OUTER join
- c. Right OUTER join
- d. Full OUTER join
- e. Self-join
- f. Natural join
- g. Cartesian products



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to :

- 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 tables



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 specifying the `CROSS JOIN` clause.

Table Aliases

- Table aliases speed up database access.
- They can help to keep SQL code smaller by conserving memory.
- They are sometimes mandatory to avoid column ambiguity.

Practice 7: Overview

This practice covers the following topics:

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using Subqueries to Solve Queries

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cornelia Sirbu (vrabie.cornelia@gmail.com) has a non-transferable
license to use this Student Guide.

Objectives

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

- Define subqueries
- Describe the types of problems that subqueries can solve
- Identify the types of subqueries
- Write single-row, multiple-row, multiple-column subqueries



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using a Subquery to Solve a Problem



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Suppose the HR manager wants a report of all employees who were hired after Davies. The HR manager submits a request for the report to the IT department.

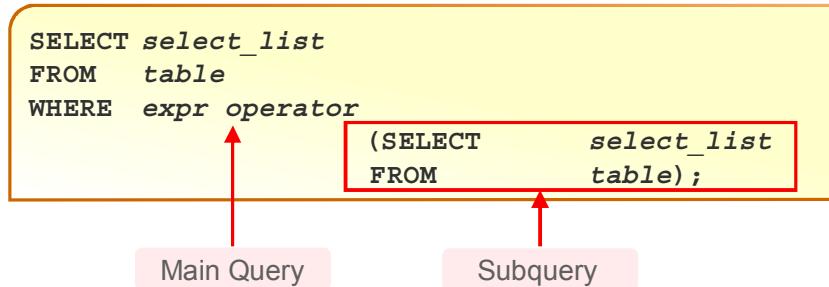
To solve this problem, the IT manager needs *two* queries: one query to find when Davies was hired, and another to find who were hired after Davies.

The IT manager 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*).

Subquery Syntax

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



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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.

A subquery is a `SELECT` statement that is embedded in the clause of another `SELECT` statement.

You can place the subquery in a number of SQL clauses, including the following:

- `WHERE` clause
- `HAVING` clause
- `FROM` clause

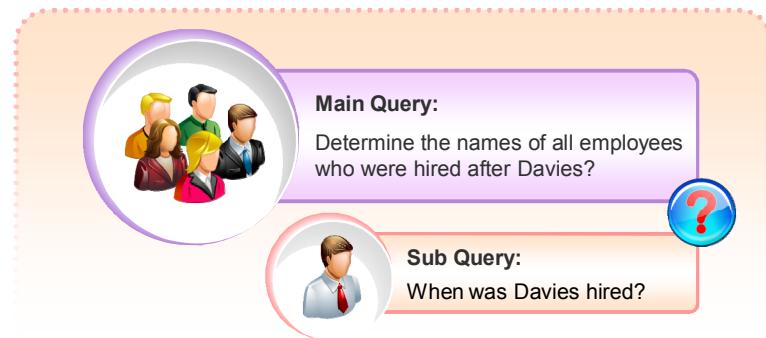
In the syntax:

`operator` includes a comparison condition such as `>`, `=`, or `IN`

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, hire_date
FROM   employees
WHERE  hire_date > (SELECT hire_date
                      FROM   employees
                      WHERE  last_name = 'Davies');
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, the inner query determines the hire date of the employee, Davies. The outer query takes the result of the inner query and uses this result to display all the employees who were hired after Davies.

Rules and 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.

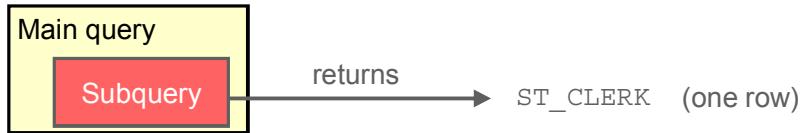


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Types of Subqueries

- Single-row subquery



- Multiple-row subquery



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- **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. They are covered in the *Oracle Database: SQL Workshop II* course.

Lesson Agenda

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



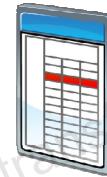
ORACLE®

Copyright © 2017, Oracle and/or its affiliates. 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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 table in the slide lists 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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary = 2500
       (SELECT MIN(salary)
        FROM   employees);
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

HAVING Clause with Subqueries

The Oracle server:

- Executes the subqueries first
- Returns the result into the HAVING clause of the main query

```
SELECT      department_id, MIN(salary)
FROM        employees
GROUP BY    department_id
HAVING      MIN(salary) > 2500
              (SELECT MIN(salary)
               FROM   employees
               WHERE  department_id = 50);
```

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
4	110	8300
5	80	8600
6	60	4200
7	10	4400

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 the minimum salary of department 30.

Another Example:

Find the job with the lowest average salary.

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

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
  FROM employees
 WHERE salary =  
      (SELECT MIN(salary)
       FROM employees
      GROUP BY department_id);
```

ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
"Cause:
"Action:

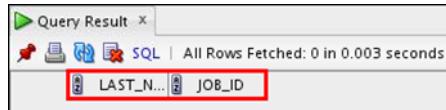
Single-row operator with multiple-row subquery



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

No Rows Returned by the Inner Query

```
SELECT last_name, job_id
  FROM employees
 WHERE job_id =
       (SELECT job_id
        FROM   jobs
       WHERE  job_title = 'Architect');
```



The subquery returns no rows because there is no job with the title "Architect."

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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
- Multiple-column subqueries
- Null values in a subquery



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. 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 =, !=, >, <, <=, >=. This returns TRUE if at least one element exists in the result set of the subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. This returns TRUE if the relation is TRUE for all elements in the result set of the subquery.

ORACLE

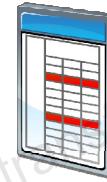
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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';
  
```

	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 © 2017, Oracle and/or its affiliates. All rights reserved.

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';
```

#	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 © 2017, Oracle and/or its affiliates. All rights reserved.

Multiple-Column Subqueries

- A multiple-column subquery returns more than one column to the outer query.
- Column comparisons in multiple column comparisons can be pairwise or nonpairwise.
- A multiple-column subquery can also be used in the `FROM` clause of a `SELECT` statement.

Syntax:

```
SELECT column, column, ...
  FROM table
 WHERE (column1, column2, ...) IN
       (SELECT column1, column2, ...
  FROM table
 WHERE condition);
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

A multiple-column subquery returns more than one column to the outer query and can be listed in the outer query's `FROM`, `WHERE`, or `HAVING` clause.

If you want to compare two or more columns, you must write a compound `WHERE` clause using logical operators. Multiple-column subqueries enable you to combine duplicate `WHERE` conditions into a single `WHERE` clause.

`IN` operator is used to check a value within a set of values. The list of values may come from the results returned by a subquery.

Syntax:

```
SELECT column, column, ...
  FROM table
 WHERE (column, column, ...) IN
       (SELECT column, column, ...
  FROM table
 WHERE condition);
```

Multiple-Column Subquery: Example

Display all the employees with the lowest salary in each department.

```
SELECT first_name, department_id, salary
FROM employees
WHERE (salary, department_id) IN
    (SELECT min(salary), department_id
     FROM employees
     GROUP BY department_id)
ORDER BY department_id;
```

#	FIRST_NAME	DEPARTMENT_ID	SALARY
1	Jennifer	10	4400
2	Pat	20	6000
3	Peter	50	2500
4	Diana	60	4200
5	Jonathon	80	8600
6	Neena	90	17000
7	Lex	90	17000
8	William	110	8300



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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

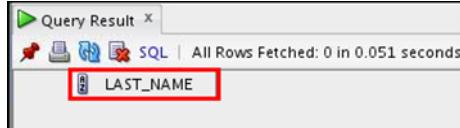


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. 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);
```



The subquery returns no rows because one of the values returned by a subquery is null.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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);
```

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.

- a. True
- b. False



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: a

Summary

In this lesson, you should have learned how to:

- Define subqueries
- Identify the types of problems that subqueries can solve
- Identify the types of subqueries
- Write single-row, multiple-row, multiple-column subqueries



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 8: 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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.

Using Set Operators

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. 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 © 2017, Oracle and/or its affiliates. All rights reserved.

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

Lesson Agenda

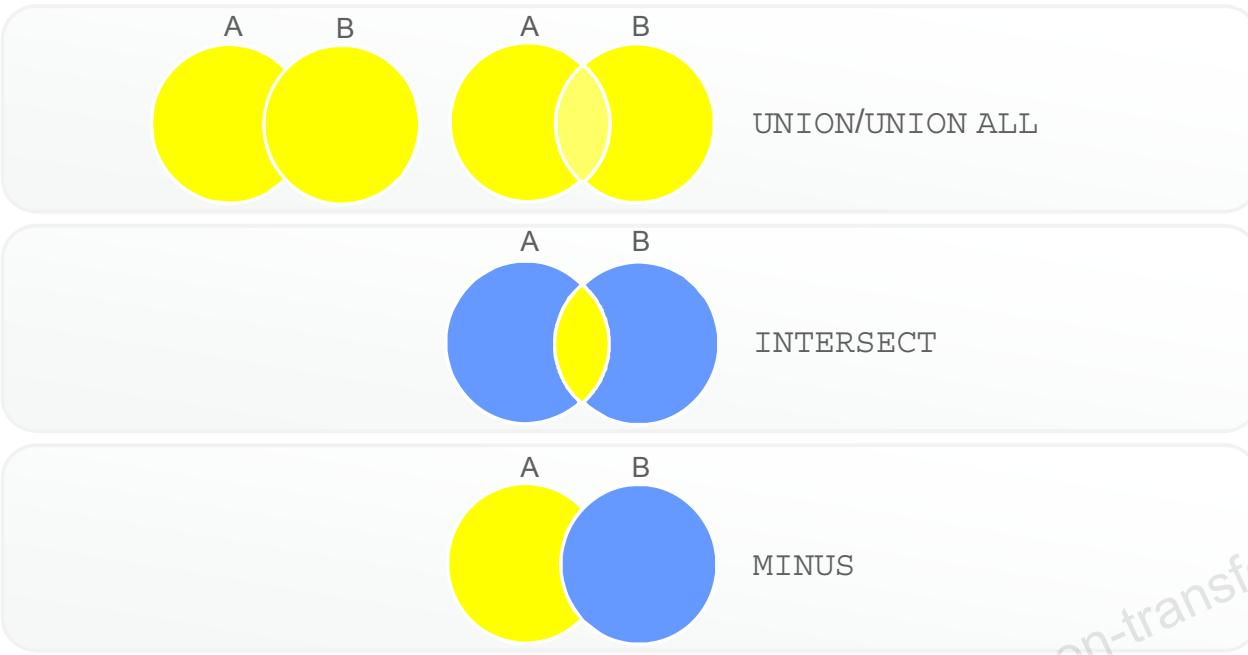
- Set operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching SELECT statements
- Using the ORDER BY clause in set operations



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Set Operators



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 duplicates
UNION ALL	Rows from both queries, including all duplicates
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 Rules

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

- The expressions in the SELECT lists of the queries must match in number and data type. When you use the UNION, UNION ALL, INTERSECT, and MINUS operators, ensure that the SELECT lists have the same number and data type of columns. The data type sometimes may not be exactly the same. In such cases, 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.
- You can use the set operators in subqueries.
- You should use parentheses to specify the order of evaluation in queries that use the INTERSECT operator with other set operators. According to SQL standards, the INTERSECT operator has 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 © 2017, Oracle and/or its affiliates. All rights reserved.

The SELECT lists of a compound query must match the corresponding expressions 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 SELECT statements
- Using the ORDER BY clause in set operations



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Tables Used in This Lesson

The tables used in this lesson are:

- EMPLOYEES: Provides details about all current employees
- RETIRED_EMPLOYEES: Provides details about all past employees



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Two tables are used in this lesson: EMPLOYEES and RETIRED_EMPLOYEES.

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.

RETIRED_EMPLOYEES stores the details of the employees who have left the company.

The structure of and data from the EMPLOYEES and RETIRED_EMPLOYEES tables are shown on the following pages.

```
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)

SELECT employee_id, last_name, job_id, hire_date, department_id
 FROM employees;

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	DEPARTMENT_ID
1	100	King	AD_PRES	17-JUN-11	90
2	101	Kochhar	AD_VP	21-SEP-09	90
3	102	De Haan	AD_VP	13-JAN-09	90
4	103	Hunold	IT_PROG	03-JAN-14	60
5	104	Ernst	IT_PROG	21-MAY-15	60
6	107	Lorentz	IT_PROG	07-FEB-15	60
7	124	Mourgos	ST_MAN	16-NOV-15	50
8	141	Rajs	ST_CLERK	17-OCT-11	50
9	142	Davies	ST_CLERK	29-JAN-13	50
10	143	Matos	ST_CLERK	15-MAR-14	50
11	144	Vargas	ST_CLERK	09-JUL-14	50
12	149	Zlotkey	SA_MAN	29-JAN-16	80
13	174	Abel	SA_REP	11-MAY-12	80
14	176	Taylor	SA_REP	24-MAR-14	80
15	178	Grant	SA_REP	24-MAY-15	(null)
16	200	Whalen	AD_ASST	17-SEP-11	10
17	201	Hartstein	MK_MAN	17-FEB-12	20
18	202	Fay	MK_REP	17-AUG-13	20
19	205	Higgins	AC_MGR	07-JUN-10	110
20	206	Gietz	AC_ACCOUNT	07-JUN-10	110

DESCRIBE retired_employees

Name	Null	Type
EMPLOYEE_ID		NUMBER(7)
FIRST_NAME		VARCHAR2(20)
LAST_NAME		VARCHAR2(20)
EMAIL		VARCHAR2(25)
RETIRED_DATE		DATE
JOB_ID		VARCHAR2(20)
SALARY		NUMBER(8,2)
MANAGER_ID		NUMBER(4)
DEPARTMENT_ID		NUMBER(6)

```
SELECT * FROM retired_employees;
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	RETIRED_DATE	JOB_ID	SALARY	MANAGER_ID	DEPARTMENT_ID
1	301	Rick	Dayle	RDAYLE	18-MAR-10	AD_PRES	8000	124	90
2	302	Meena	Rac	MRAC	21-SEP-11	AD_VP	11000	149	90
3	303	Mex	Haan	MHAAN	13-JAN-10	AD_VP	9500	149	80
4	304	Alexandera	Runold	ARUNOLD	03-JAN-11	IT_PROG	7500	124	60
5	305	Bruk	Ernst	BERNST	21-MAY-10	IT_PROG	6000	149	60
6	306	Dravid	Aust	DAUST	25-JUN-09	IT_PROG	4800	124	60
7	307	Raj	Patil	RPATIL	05-FEB-12	IT_PROG	4800	201	60
8	308	Rahul	Bose	RBOSE	17-AUG-12	FI_MGR	12008	124	100
9	309	Dany	Fav	DFAV	16-AUG-11	FI_ACCOUNT	9000	101	100
10	310	James	Ken	JKHEN	28-SEP-10	FI_ACCOUNT	8200	101	90
11	311	Shana	Garg	SGARG	30-SEP-10	FI_ACCOUNT	7700	201	100
12	312	Peter	Jois	PJOIS	07-JUN-14	FI_ACCOUNT	7800	124	100
13	313	Lui	Pops	LPOPS	07-DEC-10	FI_ACCOUNT	6900	201	100
14	314	DeL	Raph	DRAPH	07-DEC-12	PU_MAN	11000	101	30
15	315	Alex	Khurl	AKHURL	18-MAY-11	PU_CLERK	3100	149	30

Lesson Agenda

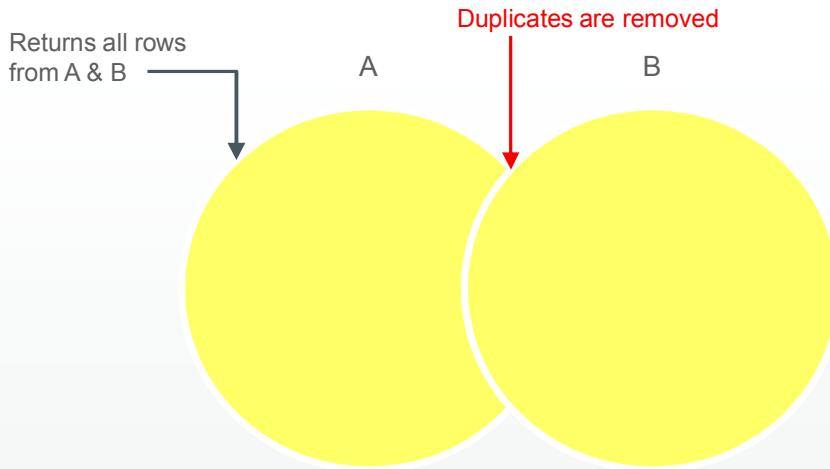
- Set operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching SELECT statements
- Using the ORDER BY clause in set operations



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

UNION Operator



The UNION operator returns rows from both queries after eliminating duplications.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 job details of all the current and retired employees. Display each job only once.

```
SELECT job_id  
FROM employees  
UNION  
SELECT job_id  
FROM retired_employees
```

JOB_ID
1 AC_ACCOUNT
2 AC_MGR
3 AD_ASST
4 AD_PRES
5 AD_VP
6 FI_ACCOUNT
7 FI_MGR
8 IT_PROG
9 MK_MAN
10 MK_REP
11 PU_CLERK
12 PU_MAN
13 SA_MAN
14 SA_REP
15 ST_CLERK
16 ST_MAN

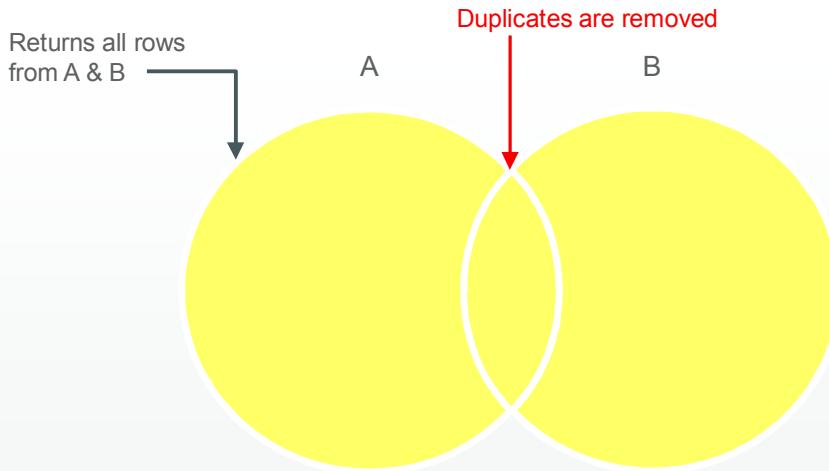


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The UNION operator eliminates any duplicate records. If records that occur in both the EMPLOYEES and the RETIRED_EMPLOYEES tables are identical, the records are displayed only once.

UNION ALL Operator



The UNION ALL operator returns rows from both queries, including all duplications.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Using the UNION ALL Operator

Display the jobs and departments of all current and previous employees.

```
SELECT job_id, department_id
FROM employees
UNION ALL
SELECT job_id, department_id
FROM retired_employees
ORDER BY job_id;
```

JOB_ID	DEPARTMENT_ID	
1 AC_ACCOUNT	110	
2 AC_MGR	110	
3 AD_ASST	10	
4 AD_PRES	90	
5 AD_PRES	90	
6 AD_VP	90	
7 AD_VP	80	
8 AD_VP	90	
9 AD_VP	90	
...		
28 SA_REP	80	
29 SA_REP	80	
30 SA_REP	(null)	
31 ST_CLERK	50	
32 ST_CLERK	50	
33 ST_CLERK	50	
34 ST_CLERK	50	
35 ST_MAN	50	



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, 35 rows are selected. The combination of the two tables totals to 35 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 job_id, department_id
FROM employees
UNION
SELECT job_id, department_id
FROM retired_employees
ORDER BY job_id;
```

The preceding query returns 19 rows. This is because it eliminates all the duplicate rows.

Lesson Agenda

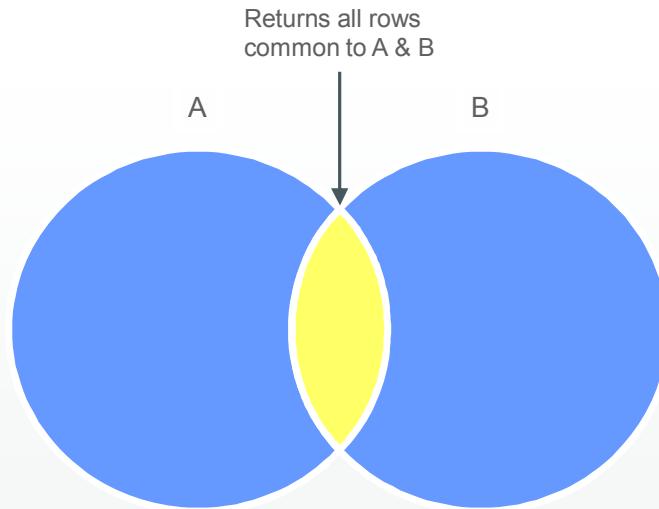
- Set operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching SELECT statements
- Using ORDER BY clause in set operations



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

INTERSECT Operator



The INTERSECT operator returns rows that are common to both queries.

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

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 common manager IDs and department IDs of current and previous employees.

```
SELECT manager_id,department_id  
FROM employees  
INTERSECT  
SELECT manager_id,department_id  
FROM retired_employees
```

	MANAGER_ID	DEPARTMENT_ID
1	149	80



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

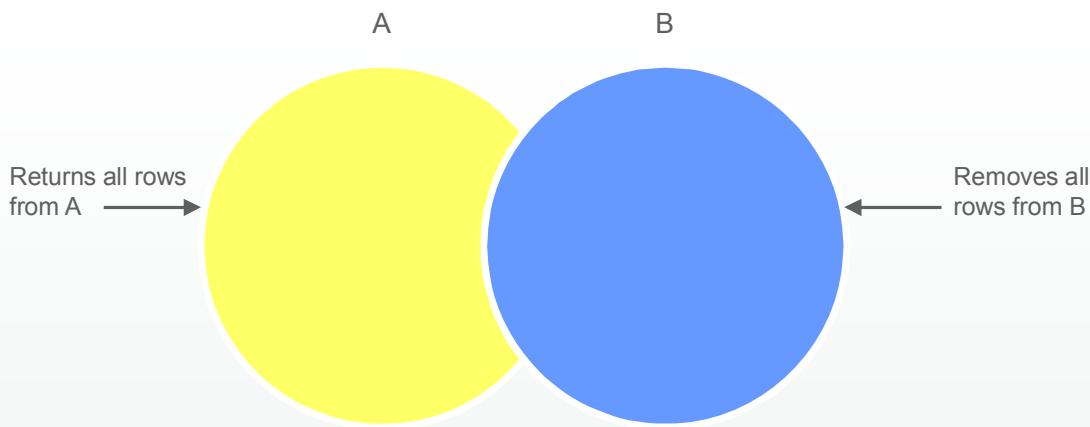
- Set operators: Types and guidelines
- Tables used in this lesson
- UNION and UNION ALL operator
- INTERSECT operator
- MINUS operator
- Matching SELECT statements
- Using the ORDER BY clause in set operations



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

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 © 2017, Oracle and/or its affiliates. All rights reserved.

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 manager IDs and Job IDs of employees whose managers have never managed retired employees in the Sales department.

```
SELECT manager_id, job_id
FROM employees
WHERE department_id = 80
MINUS
SELECT manager_id, job_id
FROM retired_employees
WHERE department_id = 80;
```

MANAGER_ID	JOB_ID
1	100 SA_MAN
2	149 SA_REP



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE

Matching SELECT Statements

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;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Matching the SELECT Statement: Example

Using the UNION operator, display the employee name, job ID, and hire date of all employees.

```
SELECT FIRST_NAME, JOB_ID, hire_date "HIRE_DATE"
FROM employees
UNION
SELECT FIRST_NAME, JOB_ID, TO_DATE(NULL) "HIRE_DATE"
FROM retired_employees;
```

FIRST_NAME	JOB_ID	HIRE_DATE
1 Alex	PU_CLERK	(null)
2 Alexander	IT_PROG	03-JAN-14
3 Alexander	IT_PROG	(null)
4 Bruce	IT_PROG	21-MAY-15
5 Bruk	IT_PROG	(null)
6 Curtis	ST_CLERK	29-JAN-13
7 Dany	FI_ACCOUNT	(null)
8 De1	PU_MAN	(null)
...		



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The EMPLOYEES and RETIRED_EMPLOYEES tables have several columns in common (for example, EMPLOYEE_ID, JOB_ID, and DEPARTMENT_ID). However, what if you want the query to display FIRST_NAME, JOB_ID, and HIRE_DATE using the UNION operator, knowing that HIRE_DATE exists only in the EMPLOYEES table?

The code example in the slide matches the FIRST_NAME and JOB_ID columns in the EMPLOYEES and RETIRED_EMPLOYEES tables. NULL is added to the RETIRED_EMPLOYEES SELECT statement to match the HIRE_DATE 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 RETIRED_EMPLOYEES table contains a NULL in the HIRE_DATE column.

Lesson Agenda

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

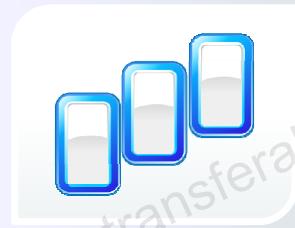


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. 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 ascending order.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the ORDER BY clause only once in a compound query. Place the ORDER BY clause 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   retired_employees
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 two set operator guidelines.

- a. The expressions in the `SELECT` lists must match in number.
- b. Parentheses cannot be used to alter the sequence of execution.
- c. The data type of each column in the second query must match the data type of its corresponding column in the first query.
- d. The `ORDER BY` clause can be used only once in a compound query, unless a `UNION ALL` operator is used.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: a, c

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 © 2017, Oracle and/or its affiliates. All rights reserved.

- 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 9: Overview

In this practice, you create reports by using:

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



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Managing Tables Using DML Statements

The ORACLE logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Objectives

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

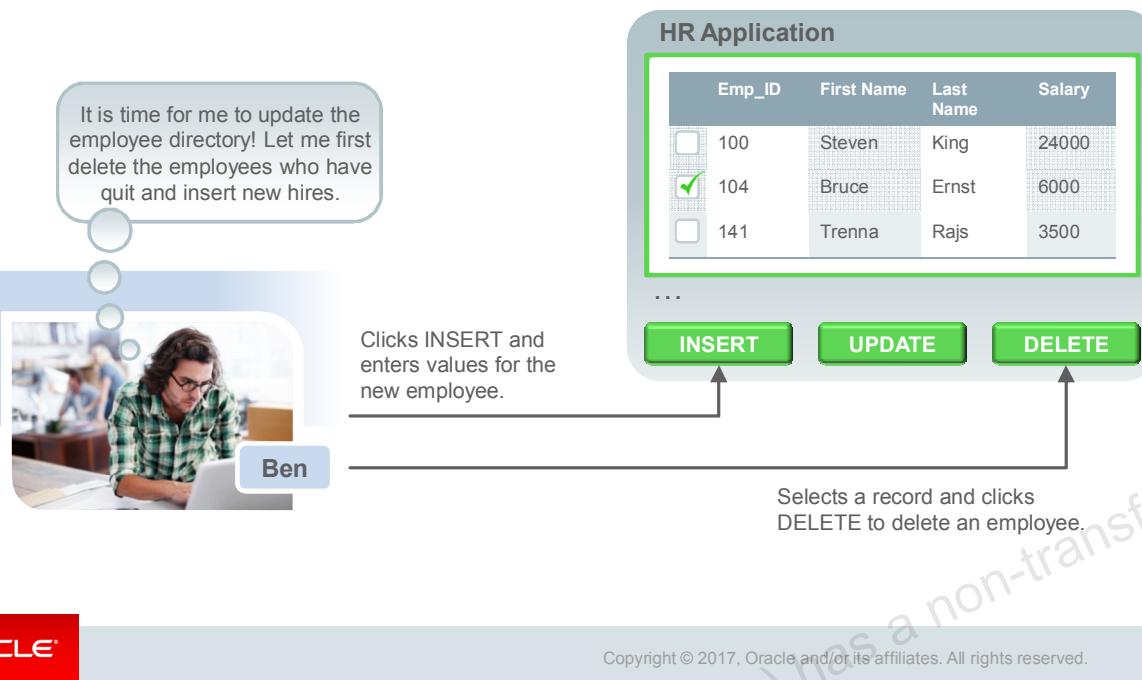
- Describe each data manipulation language (DML) statement
- Control transactions



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

HR Application Scenario



Ben is an HR manager in USA. Ben wants to update the outdated employee list in his organization because he has hired new employees recently and a few employees have left the organization.

Ben logs in to the HR application and selects the ex-employee records and clicks on **DELETE**. He then clicks **INSERT** and enters the details of new hires and clicks **SAVE**. The employee list is now updated.

When the HR manager performs these transactions in the HR application, data manipulation language (DML) statements are used in the background. DML statements modify the data in the tables. In this lesson, you learn about DML statements and how to use them.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
 - FOR UPDATE clause in a SELECT statement
 - LOCK TABLE statement

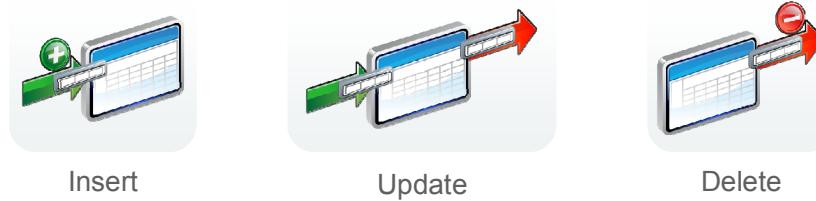


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®

Data Manipulation Language

- A DML statement is executed when you:
 - Add new rows to a table
 - Modify existing rows in a table
 - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction*.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decreasing the savings account, increasing the checking account, and recording the transaction in the transaction journal. The Oracle server must guarantee that all the three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

Note

- Most of the DML statements in this lesson assume that no constraints on the table are violated. Constraints are discussed later in this course.
- In SQL Developer, click the Run Script icon or press F5 to run the DML statements. The feedback messages will be shown on the Script Output tabbed page.

Adding a New Row to a Table

The diagram illustrates the addition of a new row to the DEPARTMENTS table. It shows two states of the table: before and after the insertion. A red arrow points from the original table to the modified table, indicating the insertion process.

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	

70 Public Relations 100 1700 New row

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	70 Public Relations	100	1700	
2	10 Administration	200	1700	
3	20 Marketing	201	1800	
4	50 Shipping	124	1500	
5	60 IT	103	1400	
6	80 Sales	149	2500	
7	90 Executive	100	1700	
8	110 Accounting	205	1700	
9	190 Contracting	(null)	1700	

Insert a new row into the DEPARTMENTS table.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The graphic in the slide illustrates the addition of a new department record to the DEPARTMENTS table.

INSERT Statement Syntax

- Add new rows to a table by using the `INSERT` statement:

```
INSERT INTO  table [(column [, column...])]
VALUES      (value [, value...]);
```

- With this syntax, only one row is inserted at a time.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can add new rows to a table by issuing the `INSERT` statement.

In the syntax:

<code>table</code>	Is the name of the table
<code>column</code>	Is the name of the column in the table to populate
<code>value</code>	Is the corresponding value for the column

Note: This statement with the `VALUES` clause adds only one row at a time to a table.

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
INSERT INTO departments(department_id,  
    department_name, manager_id, location_id)  
VALUES (70, 'Public Relations', 100, 1700);  
1 row inserted.
```

- Enclose character and date values within single quotation marks.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

```
DESCRIBE departments
```

For clarity, use the column list in the `INSERT` clause.

Enclose character and date values within single quotation marks; however, it is not recommended that you enclose numeric values within single quotation marks.

Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO departments (department_id,
                        department_name)
VALUES          (30, 'Purchasing');
1 row inserted.
```

- Explicit method: Specify the NULL keyword in the VALUES list.

```
INSERT INTO departments
VALUES          (100, 'Finance', NULL, NULL);
1 row inserted.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the NULL keyword in the VALUES list; specify the empty string (' ') in the VALUES list for character strings and dates.

Be sure that you can use null values in the targeted column by verifying the NULL status with the DESCRIBE command.

The Oracle server automatically enforces all data types, data ranges, and data integrity constraints. If a column is not explicitly listed, a null value is inserted in the new row unless you have default values for the missing columns that are used.

Common errors that can occur when you are inserting are in the following order:

- Mandatory value missing for a NOT NULL column
- Duplicate value violating any unique or primary key constraint
- Any value violating a CHECK constraint
- Referential integrity maintained for foreign key constraint
- Data type mismatches or values too wide to fit in a column

Note: Use of the column list is recommended because it makes the INSERT statement more readable and reliable, or less prone to mistakes.

Inserting Special Values

The SYSDATE function records the current date and time.

```
INSERT INTO employees (employee_id,
                      first_name, last_name,
                      email, phone_number,
                      hire_date, job_id, salary,
                      commission_pct, manager_id,
                      department_id)
VALUES
      (113,
       'Louis', 'Popp',
       'LPOPP', '515.124.4567',
       SYSDATE, 'AC_ACCOUNT', 6900,
       NULL, 205, 110);
```

1 row inserted.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Inserting Specific Date and Time Values

- Add a new employee.

```
INSERT INTO employees
VALUES      (114,
              'Den', 'Raphealy',
              'DRAPHEAL', '515.127.4561',
              TO_DATE('FEB 3, 2016', 'MON DD, YYYY'),
              'SA REP', 11000, 0.2, 100, 60);
```

1 row inserted.

- Verify your addition.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-16	SA REP	11000	0.2	100	60



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The DD-MON-RR format is generally used to insert a date value. With the RR format, the system provides the correct century automatically.

You may also supply the date value in the DD-MON-YYYY format. This is recommended because it clearly specifies the century and does not depend on the internal RR format logic of specifying the correct century.

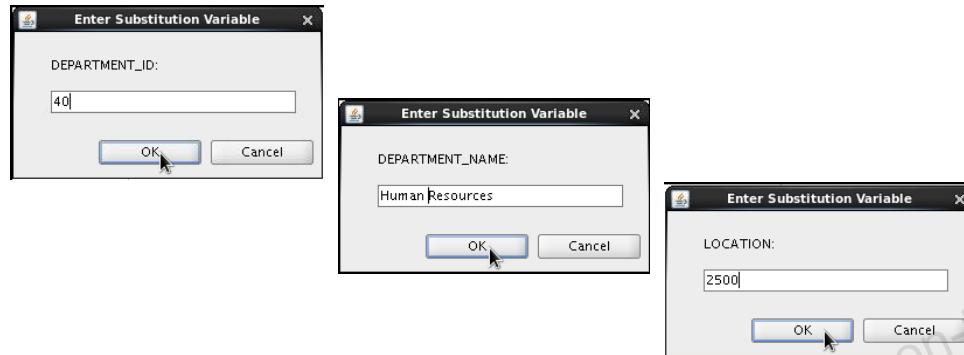
If you want to enter the date in a format other than the default format (for example, with another century or a specific time), you must use the TO_DATE function.

The example in the slide records information for employee Raphealy in the EMPLOYEES table. It sets the HIRE_DATE column to be February 3, 2003.

Creating a Script

- Use the & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
INSERT INTO departments
    (department_id, department_name, location_id)
VALUES (&department_id, '&department_name', &location);
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Copying Rows from Another Table

- Write your `INSERT` statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

5 rows inserted.

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.
- Inserts all the rows returned by the subquery in the table, `sales_reps`.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the `INSERT` statement to add rows to a table where the values are derived from existing tables. In the example in the slide, for the `INSERT INTO` statement to work, you must have already created the `sales_reps` table using the `CREATE TABLE` statement. `CREATE TABLE` is discussed in the lesson titled “Introduction to Data Definition Language.”

In place of the `VALUES` clause, you use a subquery.

Syntax

```
INSERT INTO table [ column (, column) ] subquery;
```

In the syntax:

<code>table</code>	Is the name of the table
<code>column</code>	Is the name of the column in the table to populate
<code>subquery</code>	Is the subquery that returns rows to the table

The number of columns and their data types in the column list of the `INSERT` clause must match the number of values and their data types in the subquery. Zero or more rows are added depending on the number of rows returned by the subquery. To create a copy of the rows of a table, use `SELECT *` in the subquery:

```
INSERT INTO copy_emp
SELECT *
FROM employees;
```

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
 - FOR UPDATE clause in a SELECT statement
 - LOCK TABLE statement



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

Update rows in the EMPLOYEES table:



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50



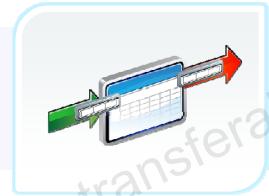
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

UPDATE Statement Syntax

- Modify existing values in a table with the UPDATE statement:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- Update more than one row at a time (if required).



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can modify the existing values in a table by using the UPDATE statement.

In the syntax:

<i>table</i>	Is the name of the table
<i>column</i>	Is the name of the column in the table to populate
<i>value</i>	Is the corresponding value or subquery for the column
<i>Condition</i>	Identifies the rows to be updated and is composed of column names, expressions, constants, subqueries, and comparison operators

Confirm the update operation by querying the table to display the updated rows.

For more information, see the section on “UPDATE” in *Oracle Database SQL Language Reference* for 12c database.

Note: In general, use the primary key column in the WHERE clause to identify a single row for update. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by last name may return more than one employee having the same last name.

Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the WHERE clause:

```
UPDATE employees  
SET department_id = 50  
WHERE employee_id = 113;
```

1 row updated.

- Values for all the rows in the table are modified if you omit the WHERE clause:

```
UPDATE copy_emp  
SET department_id = 110;  
22 rows updated
```

- Specify SET column_name= NULL to update a column value to NULL.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The UPDATE statement modifies the values of a specific row or rows if the WHERE clause is specified. The example in the slide shows the transfer of employee 113 (Popp) to department 50. If you omit the WHERE clause, values for all the rows in the table are modified. Examine the updated rows in the COPY_EMP table.

```
SELECT last_name, department_id  
FROM copy_emp;
```

For example, an employee who was an SA REP has now changed his job to an IT PROG. Therefore, his JOB_ID needs to be updated and the commission field needs to be set to NULL.

```
UPDATE employees  
SET job_id = 'IT_PROG', commission_pct = NULL  
WHERE employee_id = 114;
```

Note: The COPY_EMP table has the same data as the EMPLOYEES table.

Updating Two Columns with a Subquery

Update employee 103's job and salary to match those of employee 205.

```
UPDATE employees
SET (job_id,salary) = (SELECT job_id,salary
                         FROM employees
                           WHERE employee_id = 205)
      WHERE employee_id = 103;
```

1 row updated.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Updating Rows Based on Another Table

Use the subqueries in the UPDATE statements to update row values in a table based on values from another table:

```
UPDATE copy_emp
SET   department_id = (SELECT department_id
                        FROM employees
                        WHERE employee_id = 100)
WHERE  job_id        = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);
```

1 row updated.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
 - FOR UPDATE clause in a SELECT statement
 - LOCK TABLE statement



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®

Removing a Row from a Table

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

Delete a row from the DEPARTMENTS table:

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



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

DELETE Statement

You can remove existing rows from a table by using the `DELETE` statement:

```
DELETE [FROM]    table  
[WHERE    condition];
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the syntax:

table

Is the name of the table

condition

Identifies the rows to be deleted, and is composed of column names, expressions, constants, subqueries, and comparison operators

Note: If no rows are deleted, the message “0 rows deleted” is returned (on the Script Output tab in SQL Developer).

For more information, see the section on “`DELETE`” in *Oracle Database SQL Language Reference* for 12c database.

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
DELETE FROM departments  
WHERE department_name = 'Finance';  
1 row deleted.
```

- All rows in the table are deleted if you omit the WHERE clause:

```
DELETE FROM copy_emp;  
22 rows deleted
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can delete specific rows by specifying the WHERE clause in the DELETE statement. The first example in the slide deletes the accounting department from the DEPARTMENTS table. You can confirm the delete operation by trying to display the deleted rows using the SELECT statement. The query returns 0 rows.

```
SELECT *  
FROM departments  
WHERE department_name = 'Finance';
```

However, if you omit the WHERE clause, all rows in the table are deleted. The second example in the slide deletes all rows from the COPY_EMP table, because no WHERE clause was specified.

Example

Remove rows identified in the WHERE clause.

```
DELETE FROM employees WHERE employee_id = 114;
```

```
DELETE FROM departments WHERE department_id IN (30, 40);
```

Deleting Rows Based on Another Table

Use the subqueries in the `DELETE` statements to remove rows from a table based on values from another table:

```
DELETE FROM employees
WHERE department_id IN
    (SELECT department_id
     FROM departments
     WHERE department_name
          LIKE '%Public%');

1 row deleted.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the TRUNCATE statement to quickly remove all rows from a table or cluster efficiently. Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information. Rollback information is covered later in this lesson.

If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement. You will learn more about DDL statements and disabling constraints in the lesson titled “Introduction to Data Definition Language.”

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
 - FOR UPDATE clause in a SELECT statement
 - LOCK TABLE statement



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®

Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Transactions give you more flexibility and control when changing data, and the Oracle server ensures data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that constitute one consistent change to the data. For example, a transfer of funds between two accounts should include the debit in one account and the credit to another account of the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only one DCL statement

Database Transactions: Start and End

- Begin when the first DML SQL statement is executed
- End with one of the following events:
 - A COMMIT or ROLLBACK statement is issued.
 - A DDL or DCL statement executes (automatic commit).
 - The user exits SQL Developer or SQL*Plus.
 - The system crashes.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When does a database transaction start and end?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A COMMIT or ROLLBACK statement is issued.
- A DDL statement, such as CREATE, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and, therefore, implicitly ends a transaction.

Advantages of COMMIT and ROLLBACK Statements

Using COMMIT and ROLLBACK statements, you can:

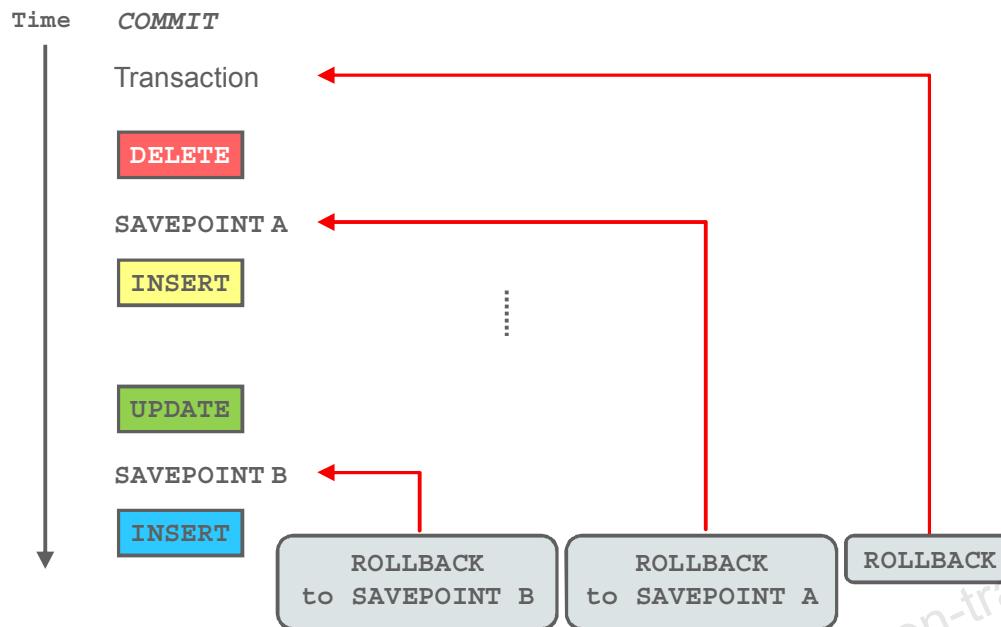
- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Explicit Transaction Control Statements



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can control the logic of transactions by using the COMMIT, SAVEPOINT, and ROLLBACK statements.

Statement	Description
COMMIT	COMMIT ends the current transaction by making all pending data changes permanent.
SAVEPOINT <i>name</i>	SAVEPOINT <i>name</i> marks a savepoint within the current transaction.
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes.
ROLLBACK TO SAVEPOINT <i>name</i>	ROLLBACK TO <savepoint> rolls back the current transaction to the specified savepoint, thereby discarding any changes and/or savepoints that were created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

Note: You cannot COMMIT to a SAVEPOINT. SAVEPOINT is not ANSI-standard SQL.

Rolling Back Changes to a Marker

- Create a marker in the current transaction by using the `SAVEPOINT` statement.
- Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```
UPDATE...
SAVEPOINT update_done; ←
SAVEPOINT update_done

INSERT...
ROLLBACK TO update_done; →
Rollback complete.
```

ROLLBACK to this point



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Implicit Transaction Processing

- An automatic commit occurs in the following circumstances:
 - A DDL statement is issued
 - A DCL statement is issued
 - A normal exit from SQL Developer or SQL*Plus, without explicitly issuing COMMIT or ROLLBACK statements
- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus, or a system failure.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Status	Circumstances
Automatic commit	A DDL statement or DCL statement is issued; SQL Developer or SQL*Plus exited normally, without explicitly issuing COMMIT or ROLLBACK commands.
Automatic rollback	Abnormal termination of SQL Developer or SQL*Plus, or system failure.

Note: In SQL*Plus, the AUTOCOMMIT command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the COMMIT statement can still be issued explicitly. Also, the COMMIT statement is issued when a DDL statement is issued or when you exit SQL*Plus. The SET AUTOCOMMIT ON/OFF command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled. To enable Autocommit, perform the following:

- From the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.
- In the right pane, select the “Autocommit in SQL Worksheet” option. Click OK.

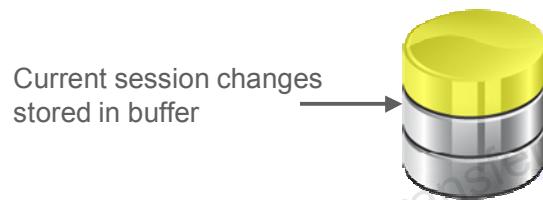
System Failures

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to the state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL*Plus, a normal exit is accomplished by entering the `EXIT` command at the prompt. Closing the window is interpreted as an abnormal exit.

State of Data Before COMMIT or ROLLBACK

- You can recover the data of the previous state.
- You can review the results of the DML operations by using the `SELECT` statement in the current session.
- Other sessions *cannot* view the results of the DML statements issued by the current session.
- The affected rows are *locked*; other sessions cannot change the data in the affected rows.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Every data change made during the transaction is temporary until the transaction is committed.

The state of the data before `COMMIT` or `ROLLBACK` statements are issued can be described as follows:

- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current session can review the results of the data manipulation operations by querying the tables.
- Other sessions cannot view the results of the data manipulation operations made by the current session. The Oracle server institutes read consistency to ensure that each session sees data as it existed at the last commit.
- The affected rows are locked; other sessions cannot change the data in the affected rows.

State of Data After COMMIT

- Data changes are saved in the database.
- The previous state of the data is overwritten.
- All sessions can view the results.
- Locks on the affected rows are released; those rows are available for other sessions to manipulate.
- All savepoints are erased.



COMMIT

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Make all pending changes permanent by using the **COMMIT** statement. The following happens after a **COMMIT** statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All sessions can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other sessions to perform new data changes.
- All savepoints are erased.

Committing Data

- Make the changes:

```
DELETE FROM employees  
WHERE employee_id = 113;  
1 row deleted.  
  
INSERT INTO departments  
VALUES (290, 'Corporate Tax', NULL, 1700);  
1 row inserted.
```

- Commit the changes:

```
COMMIT;
```

```
Commit complete.
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, a row is deleted from the EMPLOYEES table and a new row is inserted into the DEPARTMENTS table. The changes are saved by issuing the COMMIT statement.

Example

Remove departments 290 and 300 from the DEPARTMENTS table and update a row in the EMPLOYEES table. Save the data change.

```
DELETE FROM departments  
WHERE department_id IN (290, 300);  
  
UPDATE employees  
SET department_id = 80  
WHERE employee_id = 206;  
  
COMMIT;
```

State of Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;  
ROLLBACK;
```



ROLLBACK

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Discard all pending changes by using the ROLLBACK statement, which results in the following:

- Data changes are undone.
- The previous state of the data is restored.
- Locks on the affected rows are released.

State of Data After ROLLBACK: Example

```
DELETE FROM test;  
4 rows deleted.  
  
ROLLBACK;  
Rollback complete.  
  
DELETE FROM test WHERE id = 100;  
1 row deleted.  
  
SELECT * FROM test WHERE id = 100;  
No rows selected.  
  
COMMIT;  
Commit complete.
```

While attempting to remove a record from the TEST table, you may accidentally empty the table. However, you can correct the mistake by rolling back, reissue a proper statement, and make the data change permanent with COMMIT statement.

Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
 - FOR UPDATE clause in a SELECT statement
 - LOCK TABLE statement



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®

Read Consistency

- Read consistency guarantees a consistent view of data at all times.
- Changes made by one user do not conflict with the changes made by another user.
- Read consistency ensures that, on the same data:
 - Readers do not wait for writers
 - Writers do not wait for readers
 - Writers wait for writers



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Database users access the database in two ways:

- Read operations (`SELECT` statement)
- Write operations (`INSERT`, `UPDATE`, `DELETE` statements)

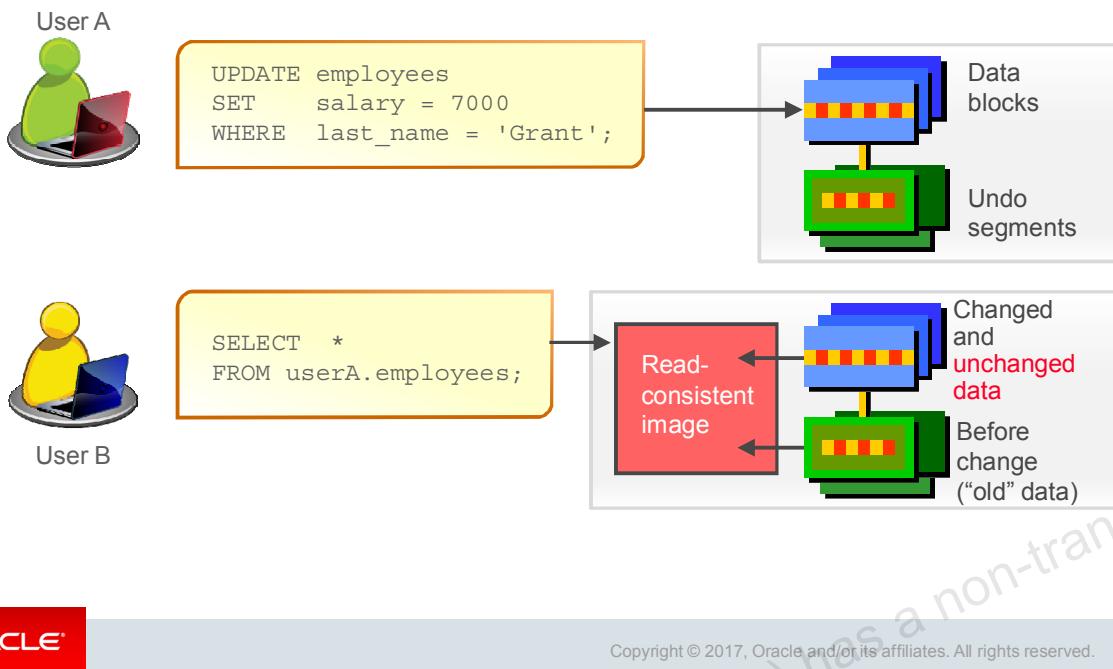
You need read consistency so that:

- The database reader and writer are ensured a consistent view of data
- Readers do not view data that is in the process of being changed
- Writers are ensured that the changes to the database are done in a consistent manner
- Changes made by one writer do not disrupt or conflict with the changes being made by another writer

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

Note: The same user can log in to different sessions. Each session maintains read consistency in the manner described above, even if they are the same users.

Implementing Read Consistency



Read consistency is an automatic implementation. It keeps a partial copy of the database in the undo segments. The read-consistent image is constructed from the committed data in the table and the old data that is being changed and is not yet committed from the undo segment.

When an insert, update, or delete operation is made on the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

Before the changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a `SELECT` statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:

- The original, older version of the data in the undo segment is written back to the table.
- All users see the database as it existed before the transaction began.

Lesson Agenda

- Adding new rows in a table
 - INSERT statement
- Changing data in a table
 - UPDATE statement
- Removing rows from a table:
 - DELETE statement
 - TRUNCATE statement
- Database transaction control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- Manual Data Locking
 - FOR UPDATE clause in a SELECT statement
 - LOCK TABLE statement



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ORACLE®

FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the EMPLOYEES table where job_id is SA_REP.

```
SELECT employee_id, salary, commission_pct, job_id
  FROM employees
 WHERE job_id = 'SA_REP'
   FOR UPDATE
 ORDER BY employee_id;
```

- Lock is released only when you issue a ROLLBACK or a COMMIT.
- If the SELECT statement attempts to lock a row that is locked by another user, the database waits until the row is available, and then returns the results of the SELECT statement.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you issue a SELECT statement against the database to query some records, no locks are placed on the selected rows. In general, this is required because the number of records locked at any given time is (by default) kept to the absolute minimum: only those records that have been changed but not yet committed are locked. Even then, others will be able to read those records as they appeared before the change (the “before image” of the data). There are times, however, when you may want to lock a set of records even before you change them in your program. Oracle offers the FOR UPDATE clause of the SELECT statement to perform this locking.

When you issue a SELECT . . . FOR UPDATE statement, RDBMS automatically obtains exclusive row-level locks on all the rows identified by the SELECT statement, thereby holding the records “for your changes only.” No one else will be able to change any of these records until you perform a ROLLBACK or a COMMIT.

You can append the optional keyword NOWAIT to the FOR UPDATE clause to tell the Oracle server not to wait if the table has been locked by another user. In this case, control will be returned immediately to your program or to your SQL Developer environment so that you can perform other work, or simply wait for a period of time before trying again. Without the NOWAIT clause, your process will block until the table is available, when the locks are released by the other user through the issue of a COMMIT or a ROLLBACK command.

FOR UPDATE Clause: Examples

- You can use the FOR UPDATE clause in a SELECT statement against multiple tables.

```
SELECT e.employee_id, e.salary, e.commission_pct
  FROM employees e JOIN departments d
    USING (department_id)
   WHERE job_id = 'ST_CLERK'
     AND location_id = 1500
    FOR UPDATE
 ORDER BY e.employee_id;
```

- Rows from both the EMPLOYEES and DEPARTMENTS tables are locked.
- Use FOR UPDATE OF *column_name* to qualify the column that you intend to change; then only the rows from that specific table are locked.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In the following example, the database is instructed to wait for five seconds for the row to become available, and then return control to you.

```
SELECT employee_id, salary, commission_pct, job_id  
FROM employees  
WHERE job_id = 'SA_REP'  
FOR UPDATE WAIT 5  
ORDER BY employee_id;
```

LOCK TABLE Statement

- Use the `LOCK TABLE` statement to lock one or more tables in a specified mode.
- This manually overrides automatic locking.
- Tables are locked until you `COMMIT` or `ROLLBACK`.

```
LOCK TABLE table_name
IN [ROW SHARE/ROW EXCLUSIVE/SHARE UPDATE/SHARE/
    SHARE ROW EXCLUSIVE/ EXCLUSIVE] MODE
[NOWAIT];
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the `LOCK TABLE` statement to manually lock the tables and override automatic locking. The table must be in your schema or you must have the `LOCK ANY TABLE` privilege.

The `lockmode` clause:

SHARE	Permits concurrent queries but prevents update on the locked table
EXCLUSIVE	Permits queries on the locked table but prevents any other activity

For more information about the other `lockmode` clauses, refer to the `LOCK TABLE` statement in *Oracle Database SQL Language Reference* for 12c database.

You can append the optional keyword `NOWAIT` to the `LOCK TABLE` statement to tell the Oracle server not to wait if the table has been locked by another user.

For example, the following statement locks the `EMPLOYEES` table in exclusive mode but does not wait if another user has already locked the table.

```
LOCK TABLE employees
IN EXCLUSIVE MODE
NOWAIT;
```

Quiz



The following statements produce the same results:

`DELETE FROM copy_emp;`

`TRUNCATE TABLE copy_emp;`

- a. True
- b. False



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: b

Summary

In this lesson, you should have learned how to use the following statements:

Function	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
TRUNCATE	Removes all rows from a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes
FOR UPDATE clause in SELECT	Locks rows identified by the SELECT query



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Practice 10: Overview

This practice covers the following topics:

- Inserting rows into the tables
- Updating and deleting rows in the table
- Controlling transactions



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In this practice, you add rows to the MY_EMPLOYEE table, update and delete data from the table, and control your transactions. You run a script to create the MY_EMPLOYEE table.

Introduction to Data Definition Language

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red rectangular background.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Cornelia Sirbu (vrabie.cornelia@gmail.com) has a non-transferable
license to use this Student Guide.

Objectives

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

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

HR Application Scenario

The JOBS table looks fine except it does not contain a column for JOB_TITLE. What should I do now?

Bob



HR Application

job_ID	Min_Salary	Max_Salary
AD_PRES	20080	40000
SA_MAN	10000	20080
IT_PROG	4000	10000

...

EDIT

HR Application

Action:	Add Column
Name:	Job Title
Datatype:	varchar2(25)
Default:	

SUBMIT

job_ID	Min_Salary	Max_Salary	Job_Title
AD_PRES	20080	40000	NULL
SA_MAN	10000	20080	NULL
IT_PROG	4000	10000	NULL



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Consider a scenario where Bob, an HR manager is creating tables to store all the employee information in the database. While creating the JOBS table to store all the job information, he forgets to create a column for the job title. So what should Bob do now?

Should he drop the table and create JOBS table again? No!

Bob can alter the table and add a new column (JOB_TITLE) to the existing JOBS table. The statements that allow you to modify the structure of database objects are called data definition language (DDL) statements. Usually, the permission to execute DDL statements is given only to the admin.

Bob submits the request to alter the JOBS table structure along with the details. The DBA receives the request and constructs an appropriate SQL statement to alter the JOBS table.

A new column called Job_Title is added to the JOBS table. The value for Job_Title remains NULL until Bob goes to the application and updates the values for all the records.

In this lesson, you learn more about DDL statements.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Database Objects

Object	Description
Table	Is the basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative name to an object

**ORACLE**

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The Oracle database can contain multiple data objects. Remember to outline each object in the database design so that it can be created during the build stage of database development.

- **Table:** Stores data
- **View:** Is a subset of data from one or more tables
- **Sequence:** Generates numeric values
- **Index:** Improves the performance of some queries
- **Synonym:** Gives an alternative name to an object

Oracle Table Structures

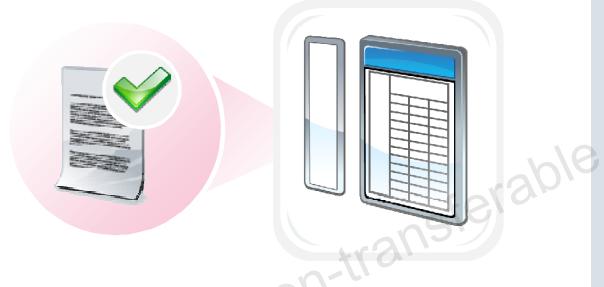
- You can create tables at any time, even when users are using the database.
- You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
- You can also modify the table structure online.

Note: More database objects are covered in the course titled *Oracle Database 12c: Workshop II*.

Naming Rules for Tables and Columns

Ensure that the table names and column names:

- Begin with a letter
- Are 1–30 characters long
- Contain only A–Z, a–z, 0–9, _, \$, and #
- Do *not* duplicate the name of another object owned by the same user
- Are *not* Oracle server–reserved words



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Name the database tables and columns according to the standard rules for naming any Oracle database object.

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, _ (underscore), \$, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must not be an Oracle server–reserved word.
 - You may also use quoted identifiers to represent the name of an object. A quoted identifier begins and ends with double quotation marks (""). If you name a schema object using a quoted identifier, you must use the double quotation marks whenever you refer to that object. Quoted identifiers can be reserved words, although this is not recommended.

Naming Guidelines

Use descriptive names for tables and other database objects.

Note: Names are not case-sensitive. For example, EMPLOYEES is treated to be the same name as eMPLOYEES or eMpLOYEES. However, quoted identifiers are case-sensitive.

For more information, see the “Schema Object Names and Qualifiers” section in the *Oracle Database SQL Language Reference* for 12c database.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

CREATE TABLE Statement

- You must have:
 - The CREATE TABLE privilege
 - A storage area

```
CREATE TABLE [schema.]table
  (column datatype [DEFAULT expr] [, ...]);
```

- You specify:
 - The table name
 - The column name, column data type, and column size



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the DDL statements that are a subset of the SQL statements used to create, modify, or remove Oracle Database structures. These statements have an immediate effect on the database and they also record information in the data dictionary. The data dictionary is an important set of read-only tables that provide database information.

To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator (DBA) uses data control language (DCL) statements to grant privileges to users.

In the syntax:

<i>schema</i>	Is the same as the owner's name
<i>table</i>	Is the name of the table
<i>DEFAULT expr</i>	Specifies a default value if a value is omitted in the INSERT statement
<i>column</i>	Is the name of the column
<i>datatype</i>	Is the column's data type and length

Note: The CREATE ANY TABLE privilege is needed to create a table in any schema other than the user's schema.

Creating Tables

- Create the table:

```
CREATE TABLE dept
  (deptno      NUMBER(2),
   dname       VARCHAR2(14),
   loc         VARCHAR2(13),
   create_date DATE DEFAULT SYSDATE);
table DEPT created.
```

- Confirm table creation:

```
DESCRIBE dept
```

```
DESCRIBE dept
Name          Null Type
-----
DEPTNO        NUMBER(2)
DNAME         VARCHAR2(14)
LOC           VARCHAR2(13)
CREATE_DATE    DATE
```

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The example in the slide creates the DEPT table with four columns: DEPTNO, DNAME, LOC, and CREATE_DATE.

The CREATE_DATE column has a default value. If a value is not provided for an INSERT statement, the system date is automatically inserted.

To confirm that the table was created, run the DESCRIBE command.

Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

Note: You can view the list of tables that you own by querying the data dictionary, as shown in the following example:

```
select table_name from user_tables;
```

Using data dictionary views, you can also find information about other database objects, such as views, indexes, and so on. You will learn about data dictionaries in detail in the *Oracle Database: SQL Workshop II* course.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Data Types

Data Type	Description
VARCHAR2 (<i>size</i>)	Variable-length character data
CHAR (<i>size</i>)	Fixed-length character data
NUMBER (<i>p, s</i>)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE).
RAW and LONG RAW	Raw binary data
BLOB	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you identify a column for a table, you need to provide a data type for the column. There are several data types available:

Data Type	Description
VARCHAR2 (<i>size</i>)	Variable-length character data (A maximum <i>size</i> must be specified; minimum <i>size</i> is 1.) Maximum size is: <ul style="list-style-type: none"> • 32767 bytes if MAX_SQL_STRING_SIZE = EXTENDED • 4000 bytes if MAX_SQL_STRING_SIZE = LEGACY
CHAR [(<i>size</i>)]	Fixed-length character data of length <i>size</i> bytes (Default and minimum <i>size</i> is 1; maximum <i>size</i> is 2,000.)
NUMBER [(<i>p, s</i>)]	Number having precision <i>p</i> and scale <i>s</i> (Precision is the total number of decimal digits and scale is the number of digits to the right of the decimal point; precision can range from 1 through 38, and scale can range from -84 through 127.)
DATE	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.

Data Type	Description
LONG	Variable-length character data (up to 2 GB)
CLOB	A character large object containing single-byte or multibyte characters. Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE); stores national character set data.
NCLOB	A character large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, both using the database national character set. Maximum size is (4 gigabytes - 1) * (database block size); stores national character set data.
RAW (size)	Raw binary data of length <code>size</code> bytes. You must specify <code>size</code> for a RAW value. Maximum <code>size</code> is: 32767 bytes if <code>MAX_SQL_STRING_SIZE</code> = EXTENDED 4000 bytes if <code>MAX_SQL_STRING_SIZE</code> = LEGACY
LONG RAW	Raw binary data of variable length up to 2 gigabytes
BLOB	A binary large object. Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	Base 64 string representing the unique address of a row in its table. This data type is primarily for values returned by the ROWID pseudocolumn.

Guidelines

- A LONG column is not copied when a table is created using a subquery.
- A LONG column cannot be included in a GROUP BY or an ORDER BY clause.
- Only one LONG column can be used per table.
- No constraints can be defined on a LONG column.
- You might want to use a CLOB column rather than a LONG column.

Datetime Data Types

You can use several datetime data types:

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes, and seconds



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Data Type	Description
TIMESTAMP	Enables storage of time as a date with fractional seconds. It stores the year, month, day, hour, minute, and the second value of the DATE data type, as well as the fractional seconds value. There are several variations of this data type, such as WITH TIMEZONE and WITH LOCALTIMEZONE.
INTERVAL YEAR TO MONTH	Enables storage of time as an interval of years and months; used to represent the difference between two datetime values in which the only significant portions are the year and month
INTERVAL DAY TO SECOND	Enables storage of time as an interval of days, hours, minutes, and seconds; used to represent the precise difference between two datetime values

Note: These datetime data types are available with Oracle9i and later releases. The datetime data types are discussed in detail in the lesson titled “Managing Data in Different Time Zones” in the *Oracle Database: SQL Workshop II* course.

Also, for more information about datetime data types, see the sections on “TIMESTAMP Datatype,” “INTERVAL YEAR TO MONTH Datatype,” and “INTERVAL DAY TO SECOND Datatype” in *Oracle Database SQL Language Reference* for 12c database.

DEFAULT Option

- Specify a default value for a column in the CREATE TABLE statement.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn is an illegal value.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates  
  (id          NUMBER(8),  
   hire_date DATE DEFAULT SYSDATE);
```

```
Table HIRE_DATES created.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you define a table, you can specify that a column should be given a default value by using the DEFAULT option. This option prevents null values from entering the columns when a row is inserted without a value for the column.

The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER); however, the value cannot be the name of another column or a pseudocolumn (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

Consider the following examples:

```
INSERT INTO hire_dates values(45, NULL);
```

The preceding statement will insert the null value rather than the default value.

```
INSERT INTO hire_dates(id) values(35);
```

The preceding statement will insert SYSDATE for the HIRE_DATE column.

Note: In SQL Developer, click the Run Script icon or press F5 to run the DDL statements. The feedback messages will be shown on the Script Output tabbed page.

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

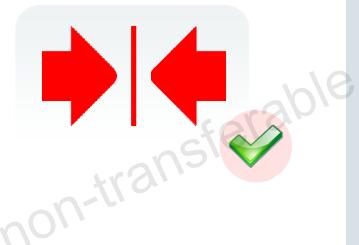


ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Including Constraints

- Constraints enforce rules at the table level.
- Constraints ensure consistency and integrity of the database.
- The following constraint types are valid:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the dropping of a table if there are dependencies from other tables.
- Provide rules for Oracle tools, such as Oracle Developer.

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a referential integrity between the column and a column of the referenced table such that values in one table match values in another table
CHECK	Specifies a condition that must be true

Constraint Guidelines

- You can name a constraint or the Oracle server generates a name by using the `SYS_Cn` format.
- Create a constraint at either of the following times:
 - At the time of table creation
 - After the creation of the table
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

All constraints are stored in the data dictionary.

Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules, except that the name cannot be the same as another object owned by the same user. If you do not name your constraint, the Oracle server generates a name with the format `SYS_Cn`, where n is an integer so that the constraint name is unique.

Constraints can be defined at the time of table creation or after the creation of the table. You can define a constraint at the column or table level. Functionally, a table-level constraint is the same as a column-level constraint.

For more information, see the section on “Constraints” in *Oracle Database SQL Language Reference* for 12c database.

Defining Constraints

- Syntax:

```
CREATE TABLE [schema.]table
  (column datatype [DEFAULT expr]
   [column_constraint],
   ...
   [table_constraint] [, ...]);
```

- Column-level constraint syntax:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint syntax:

```
column, ...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The slide gives the syntax for defining constraints when creating a table. You can create constraints at the column level or the table level.

Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition, and must refer to the column or columns to which the constraint pertains in a set of parentheses. It is mainly the syntax that differentiates the two; otherwise, functionally, a column-level constraint is the same as a table-level constraint. NOT NULL constraints can be defined only at the column level.

Constraints that apply to more than one column must be defined at the table level.

In the syntax:

schema	Is the same as the owner's name
table	Is the name of the table
DEFAULT expr	Specifies a default value to be used if a value is omitted in the INSERT statement
column	Is the name of the column
datatype	Is the column's data type and length
column_constraint	Is an integrity constraint as part of the column definition
table_constraint	Is an integrity constraint as part of the table definition

Defining Constraints: Example

- Example of a column-level constraint:

```
CREATE TABLE employees(
    employee_id  NUMBER(6)
        CONSTRAINT emp_emp_id_pk PRIMARY KEY,
    first_name    VARCHAR2(20),
    ...);
```

1

- Example of a table-level constraint:

```
CREATE TABLE employees(
    employee_id  NUMBER(6),
    first_name    VARCHAR2(20),
    ...
    job_id       VARCHAR2(10) NOT NULL,
    CONSTRAINT emp_emp_id_pk
        PRIMARY KEY (EMPLOYEE_ID));
```

2



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also be temporarily disabled.

Both examples in the slide create a primary key constraint on the EMPLOYEE_ID column of the EMPLOYEES table.

- The first example uses the column-level syntax to define the constraint.
- The second example uses the table-level syntax to define the constraint.

More details about the primary key constraint are provided later in this lesson.

NOT NULL Constraint

Ensures that null values are not permitted for the column:

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	515.123.4566	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	WGIETZ	515.123.8181	07-JUN-94

↑
NOT NULL constraint
(Primary Key enforces
NOT NULL constraint.)

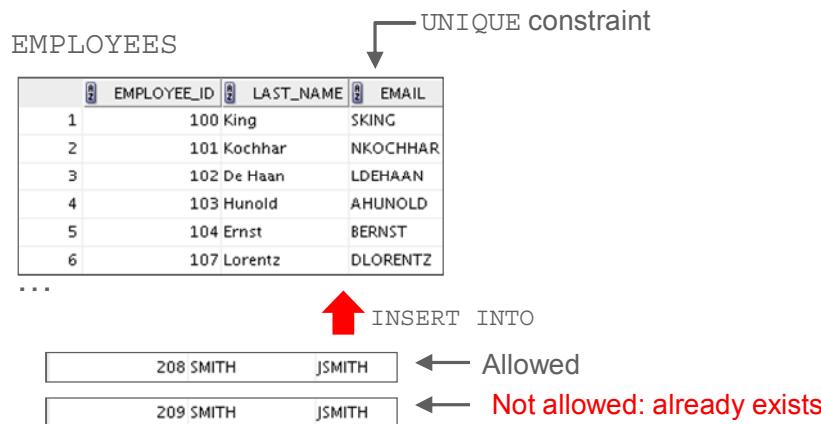
↑
NOT NULL
constraint

Absence of NOT NULL constraint (Any row
can contain a null value for this column.)



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

UNIQUE Constraint



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

A **UNIQUE** key integrity constraint requires that every value in a column or a set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or a set of columns.

The column (or set of columns) included in the definition of the **UNIQUE** key constraint is called the *unique key*. If the **UNIQUE** constraint comprises more than one column, that group of columns is called a *composite unique key*.

UNIQUE constraints enable the input of nulls unless you also define **NOT NULL** constraints for the same columns. In fact, any number of rows can include nulls for columns without the **NOT NULL** constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite **UNIQUE** key) always satisfies a **UNIQUE** constraint.

Note: Because of the search mechanism for the **UNIQUE** constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite **UNIQUE** key.

UNIQUE Constraint

Define at either the table level or the column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
    ...
    CONSTRAINT emp_email_uk UNIQUE(email));
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

PRIMARY KEY Constraint

DEPARTMENTS

PRIMARY KEY

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

INSERT INTO

(null)	Public Accounting	124	2500
50	Finance	124	1500

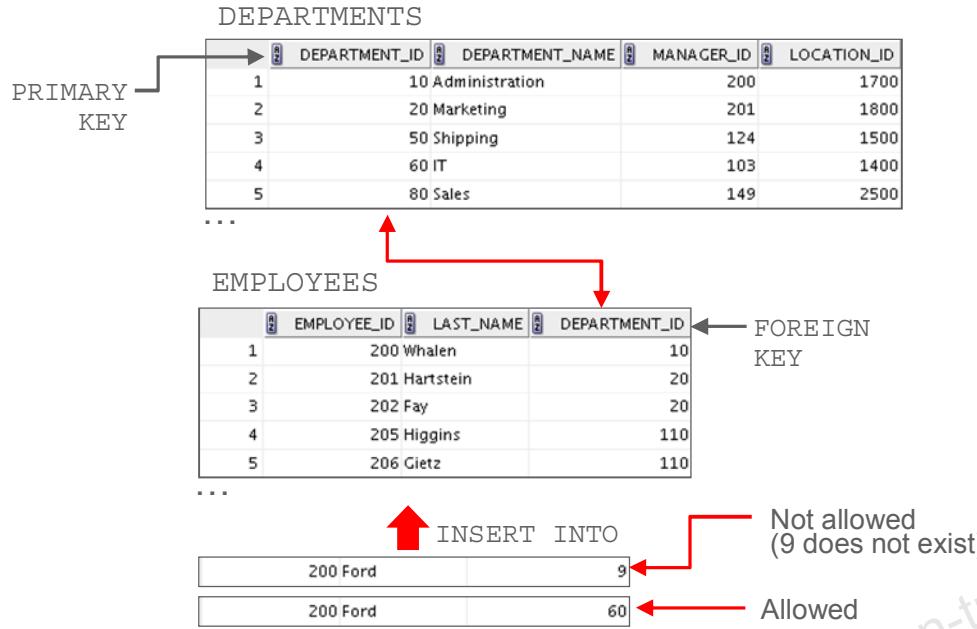
Not allowed (null value)

Not allowed (50 already exists)

ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

FOREIGN KEY Constraint



ORACLE

The FOREIGN KEY (or referential integrity) constraint designates a column or a combination of columns as a foreign key, and establishes a relationship with a primary key or a unique key in the same table or a different table.

In the example in the slide, DEPARTMENT_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT_ID column of the DEPARTMENTS table (the referenced or parent table).

Guidelines

- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

FOREIGN KEY Constraint

Define at either the table level or the column level:

```
CREATE TABLE employees (
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
    ...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
        REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.

The example in the slide defines a FOREIGN KEY constraint on the DEPARTMENT_ID column of the EMPLOYEES table, using table-level syntax. The name of the constraint is EMP_DEPT_FK.

The foreign key can also be defined at the column level, provided that the constraint is based on a single column. The syntax differs in that the keywords FOREIGN KEY do not appear, as shown in the following example:

```
CREATE TABLE employees
(
    ...
    department_id NUMBER(4) CONSTRAINT emp_deptid_fk
        REFERENCES departments(department_id),
    ...
)
```

FOREIGN KEY Constraint: Keywords

- FOREIGN KEY: Defines the column in the child table at the table-constraint level
- REFERENCES: Identifies the table and column in the parent table
- ON DELETE CASCADE: Deletes the dependent rows in the child table when a row in the parent table is deleted
- ON DELETE SET NULL: Converts dependent foreign key values to null



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The foreign key is defined in the child table and the column it references is in the parent table. The foreign key is defined using a combination of the following keywords:

- FOREIGN KEY is used to define the column in the child table at the table-constraint level.
- REFERENCES identifies the table and the column in the parent table.
- ON DELETE CASCADE indicates that when a row in the parent table is deleted, the dependent rows in the child table are also deleted.
- ON DELETE SET NULL indicates that when a row in the parent table is deleted, the foreign key values are set to null.

The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the ON DELETE CASCADE or the ON DELETE SET NULL options, the row in the parent table cannot be deleted if it is referenced in the child table. Also, these keywords cannot be used in column-level syntax.

CHECK Constraint

- Defines a condition that each row must satisfy
- Cannot reference columns from other tables

```
...., salary NUMBER(2)
CONSTRAINT emp_salary_min
    CHECK (salary > 0),...
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you define a CHECK constraint on a column, each row satisfies the condition. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null). The condition can use the same constructs as the query conditions; however, they must not refer to other values in other rows.

A single column can have multiple CHECK constraints that refer to the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

CHECK constraints can be defined at the column level or table level.

```
CREATE TABLE employees
(
    ...
    salary NUMBER(8,2) CONSTRAINT emp_salary_min
        CHECK (salary > 0),
    ...
)
```

CREATE TABLE: Example

```
CREATE TABLE teach_emp (
    empno      NUMBER(5) PRIMARY KEY,
    ename      VARCHAR2(15) NOT NULL,
    job        VARCHAR2(10),
    mgr        NUMBER(5),
    hiredate   DATE DEFAULT (sysdate),
    photo      BLOB,
    sal         NUMBER(7,2),
    deptno     NUMBER(3) NOT NULL
        CONSTRAINT admin_dept_fkey
        REFERENCES
            departments(department_id));
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Violating Constraints

```
UPDATE employees  
SET department_id = 55  
WHERE department_id = 110;
```

```
Error starting at line : 1 in command -  
UPDATE employees  
      SET department_id = 55  
      WHERE department_id = 110  
Error report -  
SQL Error: ORA-02291: integrity constraint (TEACH_A.EMP_DEPT_FK) violated - parent key not found  
02291. 00000 - "integrity constraint (%.%) violated - parent key not found"  
*Cause:   A foreign key value has no matching primary key value.  
*Action:  Delete the foreign key or add a matching primary key.
```

Department 55 does not exist.



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

When you have constraints in place on columns, an error is returned if you try to violate the constraint rule. For example, if you try to update a record with a value that is tied to an integrity constraint, an error is returned.

In the example in the slide, department 55 does not exist in the parent table, DEPARTMENTS, and therefore, you receive the “parent key not found” violation ORA- 02291.

Violating Constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments  
WHERE department_id = 60;
```

```
Error starting at line : 1 in command -  
DELETE FROM departments  
      WHERE department_id = 60  
Error report -  
SQL Error: ORA-02292: integrity constraint (TEACH_A.EMP_DEPT_FK) violated - child record found  
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"  
*Cause:    attempted to delete a parent key value that had a foreign  
           dependency.  
*Action:   delete dependencies first then parent or disable constraint.
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Lesson Agenda

- Database objects
 - Naming rules
- Data types
- CREATE TABLE statement
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Creating a Table Using a Subquery

- Create a table and insert rows by combining the CREATE TABLE statement and the AS *subquery* option.

```
CREATE TABLE table
  [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

A second method for creating a table is to apply the AS *subquery* clause, which both creates the table and inserts rows returned from the subquery.

In the syntax:

<i>table</i>	Is the name of the table
<i>column</i>	Is the name of the column, default value, and integrity constraint
<i>subquery</i>	Is the SELECT statement that defines the set of rows to be inserted into the new table

Guidelines

- The table is created with the specified column names, and the rows retrieved by the SELECT statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery SELECT list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The column data type definitions and the NOT NULL constraint are passed to the new table. Note that only the explicit NOT NULL constraint will be inherited. The PRIMARY KEY column will not pass the NOT NULL feature to the new column. Any other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

Creating a Table Using a Subquery

```
CREATE TABLE dept80
AS
SELECT employee_id, last_name,
       salary*12 ANNSAL,
       hire_date
  FROM employees
 WHERE department_id = 80;
```

Table DEPT80 created.

DESCRIBE dept80

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The example in the slide creates a table named DEPT80, which contains details of all the employees working in department 80. Notice that the data for the DEPT80 table comes from the EMPLOYEES table.

You can verify the existence of a database table and check the column definitions by using the DESCRIBE command.

However, be sure to provide a column alias when selecting an expression. The expression SALARY*12 is given the alias ANNSAL. Without the alias, the following error is generated:

```
Error starting at line 1 in command:
CREATE TABLE dept80
AS
SELECT employee_id, last_name,
       salary*12,
       hire_date
  FROM employees
 WHERE department_id = 80
Error at Command Line:4 Column:18
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 - "must name this expression with a column alias"
*Cause:
*Action:
```

Lesson Agenda

- Database objects
 - Naming rules
- Data types
- CREATE TABLE statement
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change table to read-only status



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

After you create a table, you may need to change the table structure for any of the following reasons:

- You omitted a column.
- Your column definition or its name needs to be changed.
- You need to remove columns.
- You want to put the table into read-only mode

You can do this by using the ALTER TABLE statement.

ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD      (column datatype [DEFAULT expr]
           [, column datatype] ...);
```

```
ALTER TABLE table
MODIFY   (column datatype [DEFAULT expr]
           [, column datatype] ...);
```

```
ALTER TABLE table
DROP (column [, column] ...);
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Adding a Column

- You use the ADD clause to add columns:

```
ALTER TABLE dept80
ADD          (job_id VARCHAR2(9));
Table DEPT80 altered.
```

- The new column becomes the last column:

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	149 Zlotkey	126000	29-JAN-16	(null)	
2	174 Abel	132000	11-MAY-12	(null)	
3	176 Taylor	103200	24-MAR-14	(null)	
4	206 Gietz	99600	07-JUN-10	(null)	



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Guidelines for Adding a Column

- You can add or modify columns.
- You cannot specify where the column is to appear. The new column becomes the last column.

The example in the slide adds a column named `JOB_ID` to the `DEPT80` table. The `JOB_ID` column becomes the last column in the table.

Note: If a table already contains rows when a column is added, the new column is initially null or takes the default value for all the rows. You can add a mandatory `NOT NULL` column to a table that already contains data in the other columns only if you specify a default value. You can add a `NOT NULL` column to an empty table without the default value.

Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE dept80
MODIFY      (last_name VARCHAR2(30));
Table DEPT80 altered.
```

Size of the last_name
column is modified.

- A change to the default value of a column affects only subsequent insertions to the table.



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Dropping a Column

Use the `DROP COLUMN` clause to drop columns that you no longer need from the table:

```
ALTER TABLE dept80
DROP (job_id);
```

Table DEPT80 altered.

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Ziotkey	126000	29-JAN-16
2	174	Abel	132000	11-MAY-12
3	176	Taylor	103200	24-MAR-14
4	206	Gietz	99600	07-JUN-10



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can drop a column from a table by using the `ALTER TABLE` statement with the `DROP COLUMN` clause.

Guidelines

- The column may or may not contain data.
- Using the `ALTER TABLE DROP COLUMN` statement, only one column can be dropped at a time.
- The table must have at least one column remaining in it after it is altered.
- After a column is dropped, it cannot be recovered.
- A primary key that is referenced by another column cannot be dropped, unless the cascade option is added.
- Dropping a column can take a while if the column has a large number of values. In this case, it may be better to set it to be unused and drop it when there are fewer users on the system to avoid extended locks.

Note: Certain columns can never be dropped, such as columns that form part of the partitioning key of a partitioned table or columns that form part of the `PRIMARY KEY` of an index-organized table. For more information about index-organized tables and partitioned tables, refer to the *Oracle Database Concepts* and *Oracle Database Administrator's Guide*.

SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.
- You can specify the ONLINE keyword to indicate that DML operations on the table will be allowed while marking the column or columns UNUSED.

```
ALTER TABLE      <table_name>
SET    UNUSED(<column_name> [, <column_name>]);
OR
```

```
ALTER TABLE      <table_name>
SET    UNUSED COLUMN <column_name> [, <column_name>];
```

```
ALTER TABLE <table_name>
DROP    UNUSED COLUMNS;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can use the SET UNUSED option to mark one or more columns as unused so that they can be dropped when the demand on system resources is lower. Specifying this clause does not actually remove the target columns from each row in the table (that is, it does not restore the disk space used by these columns). Therefore, the response time is faster than if you executed the DROP clause.

Unused columns are treated as if they were dropped, even though their column data remains in the table's rows. After a column has been marked as unused, a SELECT * query will not retrieve data from unused columns. In addition, the names and types of columns marked as unused will not be displayed during a DESCRIBE statement, and you can add to the table a new column with the same name as an unused column. The SET UNUSED information is stored in the USER_UNUSED_COL_TABS dictionary view.

You can specify the ONLINE keyword to indicate that DML operations on the table will be allowed while marking the column or columns UNUSED. The code example shows the use of SET UNUSED COLUMN that sets a column unused forever using the ONLINE keyword.

```
ALTER TABLE dept80 SET UNUSED(hire_date) ONLINE;
```

Note: The guidelines for setting a column to be UNUSED are similar to those for dropping a column.

DROP UNUSED COLUMNS Option

DROP UNUSED COLUMNS removes from the table all columns that are currently marked as unused. You can use this statement when you want to reclaim the extra disk space from the unused columns in the table. If the table contains no unused columns, the statement returns with no errors.

```
ALTER TABLE dept80  
SET UNUSED (last_name);
```

```
ALTER TABLE dept80  
DROP UNUSED COLUMNS;
```

Note: A subsequent DROP UNUSED COLUMNS will physically remove all unused columns from a table, similar to a DROP COLUMN.

Read-Only Tables

You can use the ALTER TABLE syntax to:

- Put a table in read-only mode, which prevents DDL or DML changes during table maintenance
- Put the table back into read/write mode

```
ALTER TABLE employees READ ONLY;  
-- perform table maintenance and then  
-- return table back to read/write mode  
  
ALTER TABLE employees READ WRITE;
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You can specify READ ONLY to convert a table into read-only mode. When the table is in READ ONLY mode, you cannot issue any DML statements that affect the table or any SELECT . . . FOR UPDATE statements. You can issue DDL statements as long as they do not modify any data in the table. Operations on indexes associated with the table are allowed when the table is in READ ONLY mode. Specify READ/WRITE to return a read-only table to read/write mode.

Note: You can drop a table that is in READ ONLY mode. The DROP command is executed only in the data dictionary, so access to the table contents is not required. The space used by the table will not be reclaimed until the tablespace is made read/write again, and then the required changes can be made to the block segment headers, and so on.

For information about the ALTER TABLE statement, see the course titled *Oracle Database 12c: SQL Workshop II*.

Lesson Agenda

- Database objects
 - Naming rules
- Data types
- CREATE TABLE statement
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Dropping a Table

- Moves a table to the recycle bin
- Removes the table and all its data entirely if the PURGE clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept80;
```

```
Table DEPT80 dropped.
```



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The `DROP TABLE` statement moves a table to the recycle bin or removes the table and all its data from the database entirely. Unless you specify the `PURGE` clause, the `DROP TABLE` statement does not result in space being released back to the tablespace for use by other objects, and the space continues to count toward the user's space quota. Dropping a table invalidates the dependent objects and removes object privileges on the table.

When you drop a table, the database loses all the data in the table and all the indexes associated with it.

Syntax

```
DROP TABLE table [PURGE]
```

In the syntax, *table* is the name of the table.

Guidelines

- All data is deleted from the table.
- Any views and synonyms remain, but are invalid.
- Any pending transactions are committed.
- Only the creator of the table or a user with the `DROP ANY TABLE` privilege can remove a table.

Note: Use the `FLASHBACK TABLE` statement to restore a dropped table from the recycle bin. This is discussed in detail in the course titled *Oracle Database 12c: SQL Workshop II*.

Quiz



Identify three actions that you perform by using constraints.

- a. Enforce rules on the data in a table whenever a row is inserted, updated, or deleted.
- b. Prevent the dropping of a table.
- c. Prevent the creation of a table.
- d. Prevent the creation of data in a table.



ORACLE®

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Answer: a, b, d

Summary

In this lesson, you should have learned how to use the CREATE TABLE, ALTER TABLE, and DROP TABLE statement to create a table, modify a table and columns, and include constraints.

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

In this lesson, you should have learned the following:

CREATE TABLE

- Use the CREATE TABLE statement to create a table and include constraints.
- Create a table based on another table by using a subquery.

DROP TABLE

- Remove rows and a table structure.
- When executed, this statement cannot be rolled back.

Practice 11: Overview

This practice covers the following topics:

- Creating new tables
- Creating a new table by using the CREATE TABLE AS syntax
- Verifying that tables exist
- Altering tables
- Adding columns
- Dropping columns
- Setting a table to read-only status
- Dropping tables



ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

You create new tables by using the CREATE TABLE statement and confirm that the new table was added to the database. You also learn to set the status of a table as READ ONLY, and then revert to READ/WRITE.

Note: For all DDL and DML statements, click the Run Script icon (or press F5) to execute the query in SQL Developer. Thus, you get to see the feedback messages on the Script Output tabbed page. For SELECT queries, continue to click the Execute Statement icon or press F9 to get the formatted output on the Results tabbed page.

