ORACLE
University

Oracle Database 19c: SQL Workshop

Student Guide – Volume I

D108644GC10

Official Internal Oracle Academy Use Only

Learn more from Oracle University at ***education.oracle.com***

Authors
Don E Bates
Shilpa Sharma
Anthony Skrabak
Apoorva Srinivas

Technical Contributors and Reviewers
Nancy Greenberg
Tulika Das
Jeremy Smyth
Purjanti Chang
Tamal Chatterjee

Publishers
Sujatha Nagendra
Pavithran Adka
Sumesh Koshy

1002222022

Copyright © 2022, Oracle and/or its affiliates.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. 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 and is not warranted to be error-free. If you find any errors, please report them to us in writing.

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 END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

Trademark Notice

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

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

Third-Party Content, Products, and Services Disclaimer

This documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Oracle Internal & Oracle Academy Use Only

Contents

1 Introduction
Objectives 1-2
Lesson Agenda 1-3
Course Objectives 1-4
Icons Used in This Course 1-5
Course Roadmap 1-6
Appendixes and Practices Used in the Course 1-13
Lesson Agenda 1-14
Oracle Database 19c: Focus Areas 1-15
Oracle Database 19c 1-16
MySQL: A Modern Database for the Digital Age 1-18
High Scalability with MySQL 1-19
MySQL-Supported Operating Systems 1-20
MySQL Enterprise Edition 1-21
Why MySQL Enterprise Edition? 1-22
Oracle Premier Support for MySQL 1-23
MySQL and Oracle Integration 1-24
Lesson Agenda 1-25
Relational and Object Relational Database Management Systems 1-26
Data Storage on Different Media 1-27
Relational Database Concept 1-28
Definition of a Relational Database 1-29
Data Models 1-30
Entity Relationship Model 1-31
Entity Relationship Modeling Conventions 1-32
Relating Multiple Tables 1-34
Relational Database Terminology 1-35
Lesson Agenda 1-36
Human Resources (HR) Application 1-37
Tables Used in This Course 1-38
Tables Used in the Course 1-39
Lesson Agenda 1-40
Using SQL to Query Your Database 1-41
How SQL Works 1-42
SQL Statements Used in the Course 1-43

Oracle Internal & Oracle Academy Use Only

Development Environments for SQL in Oracle 1-44
Introduction to Oracle Live SQL 1-45
Development Environments for SQL in MySQL 1-46
Lesson Agenda 1-47
Oracle Database Documentation 1-48
Additional Resources for Oracle 1-49
Oracle University: Oracle SQL Training 1-50
Oracle SQL Certification 1-51
MySQL Websites 1-52
MySQL Community Resources 1-53
Oracle University: MySQL Training 1-54
MySQL Certification 1-55
Summary 1-56
Practice 1: Overview 1-57

2 Retrieving Data Using the SQL SELECT Statement
Course Roadmap 2-2
Objectives 2-3
Lesson Agenda 2-4
HR Application Scenario 2-5
Writing SQL Statements 2-6
Basic SELECT Statement 2-7
Selecting All Columns 2-8
Executing SQL Statements with Oracle SQL Developer and SQL*Plus 2-9
Column Heading Defaults in SQL Developer and SQL*Plus 2-10
Executing SQL Statements in MySQL Workbench 2-11
Executing SQL Statements in mysql Command-line Client 2-12
Selecting Specific Columns 2-13
Selecting from dual with Oracle Database 2-14
Selecting Constant Expressions in MySQL 2-15
Lesson Agenda 2-16
Arithmetic Expressions 2-17
Using Arithmetic Operators 2-18
Operator Precedence 2-19
Defining a Null Value 2-20
Null Values in Arithmetic Expressions 2-21
Lesson Agenda 2-22
Defining a Column Alias 2-23
Using Column Aliases 2-24
Lesson Agenda 2-25
Concatenation Operator in Oracle 2-26

Oracle Internal & Oracle Academy Use Only

Concatenation Function in MySQL – CONCAT() 2-27
Literal Character Strings 2-28
Using Literal Character Strings in Oracle 2-29
Using Literal Character Strings in MySQL 2-30
Alternative Quote (q) Operator in Oracle 2-31
Including a Single Quotation Mark in a String with an Escape Sequence in MySQL 2-32
Duplicate Rows 2-33
Lesson Agenda 2-34
Displaying Table Structure by Using the DESCRIBE Command 2-35
Displaying Table Structure by Using Oracle SQL Developer 2-36
Displaying Table Structure by Using MySQL Workbench 2-37
Summary 2-38
Practice 2: Overview 2-39

3 Restricting and Sorting Data
Course Roadmap 3-2
Objectives 3-3
Lesson Agenda 3-4
Limiting Rows by Using a Selection 3-5
Limiting Rows That Are Selected 3-6
Using the WHERE Clause 3-7
Character Strings and Dates 3-8
Comparison Operators 3-9
Using Comparison Operators 3-10
Range Conditions Using the BETWEEN Operator 3-11
Using the IN Operator 3-12
Pattern Matching Using the LIKE Operator 3-13
Combining Wildcard Symbols 3-14
Using NULL Conditions 3-15
Defining Conditions Using Logical Operators 3-16
Using the AND Operator 3-17
Using the OR Operator 3-18
Using the NOT Operator 3-19
Lesson Agenda 3-20
Rules of Precedence 3-21
Lesson Agenda 3-23
Using the ORDER BY Clause 3-24
Sorting 3-25
Lesson Agenda 3-27
SQL Row Limiting Clause 3-28

Oracle Internal & Oracle Academy Use Only

Using SQL Row Limiting Clause in a Query in Oracle 3-29
SQL Row Limiting Clause: Example in Oracle 3-30
Using SQL Row Limiting Clause in a Query in MySQL 3-31
SQL Row Limiting Clause: Example in MySQL 3-32
Lesson Agenda 3-33
Substitution Variables in Oracle 3-34
Using the Single-Ampersand Substitution Variable 3-36
Character and Date Values with Substitution Variables 3-38
Specifying Column Names, Expressions, and Text 3-39
Using the Double-Ampersand Substitution Variable 3-40
Using the Ampersand Substitution Variable in SQL*Plus 3-41
Lesson Agenda 3-42
Using the DEFINE Command in Oracle 3-43
Using the VERIFY Command in Oracle 3-44
Using the SET Statement in MySQL 3-45
Summary 3-46
Practice 3: Overview 3-47

4 Using Single-Row Functions to Customize Output
Course Roadmap 4-2
Objectives 4-3
HR Application Scenario 4-4
Lesson Agenda 4-5
SQL Functions 4-6
Two Types of SQL Functions 4-7
Single-Row Functions 4-8
Lesson Agenda 4-10
Character Functions 4-11
Case-Conversion Functions 4-13
Using Case-Conversion Functions in WHERE Clauses in Oracle 4-14
Case-Insensitive Queries in MySQL 4-15
Character-Manipulation Functions 4-16
Using Character-Manipulation Functions 4-17
Lesson Agenda 4-18
Nesting Functions 4-19
Nesting Functions: Example 4-20
Lesson Agenda 4-21
Numeric Functions 4-22
Using the ROUND Function 4-23
Using the TRUNC Function in Oracle 4-24
Using the TRUNCATE Function in MySQL 4-25

Oracle Internal & Oracle Academy Use Only

Using the MOD Function 4-26
Lesson Agenda 4-27
Working with Dates in Oracle Databases 4-28
RR Date Format in Oracle 4-29
Using the SYSDATE Function in Oracle 4-30
Using the CURRENT_DATE and CURRENT_TIMESTAMP Functions in Oracle 4-31
Arithmetic with Dates in Oracle 4-32
Using Arithmetic Operators with Dates in Oracle 4-33
Lesson Agenda 4-34
Working with Dates in MySQL Databases 4-35
Displaying the Current Date in MySQL 4-36
Lesson Agenda 4-37
Date-Manipulation Functions in Oracle 4-38
Using Date Functions in Oracle 4-39
Using ROUND and TRUNC Functions with Dates in Oracle 4-40
Date-Manipulation Functions in MySQL 4-41
Using Date Functions in MySQL 4-42
Extracting the Month or Year Portion of Dates in MySQL 4-43
Summary 4-44
Practice 4: Overview 4-45

5 Using Conversion Functions and Conditional Expressions
Course Roadmap 5-2
Objectives 5-3
Lesson Agenda 5-4
Conversion Functions 5-5
Implicit Data Type Conversion of Strings to Numbers 5-6
Implicit Data Type Conversion of Numbers to Strings 5-7
Lesson Agenda 5-8
Using the TO_CHAR Function with Dates 5-9
Elements of the Date Format Model 5-10
Using the TO_CHAR Function with Dates 5-13
Using the TO_CHAR Function with Numbers 5-14
Using the TO_NUMBER and TO_DATE Functions 5-17
Using TO_CHAR and TO_DATE Functions with the RR Date Format 5-19
Lesson Agenda 5-20
Using the CAST() function in Oracle 5-21
Explicit Data Type Conversion of Strings to Numbers in MySQL 5-22
Explicit Data Type Conversion of Numbers to Strings in MySQL 5-23
Lesson Agenda 5-24

Oracle Internal & Oracle Academy Use Only

General Functions 5-25
NVL Function (Oracle) and IFNULL() Function (MySQL) 5-26
Using the NVL Function in Oracle 5-27
Using the NVL2 Function in Oracle 5-28
Using the IFNULL Function in MySQL 5-29
Using the NULLIF Function 5-30
Using the COALESCE Function 5-31
Lesson Agenda 5-33
Conditional Expressions 5-34
CASE Expression 5-35
Using the CASE Expression 5-36
Searched CASE Expression 5-37
DECODE Function in Oracle 5-38
Using the DECODE Function 5-39
Lesson Agenda 5-41
JSON_QUERY Function 5-42
JSON_TABLE Function 5-43
JSON_VALUE Function 5-44
Summary 5-45
Practice 5: Overview 5-46

6 Reporting Aggregated Data Using the Group Functions
Course Roadmap 6-2
Objectives 6-3
Lesson Agenda 6-4
Group Functions 6-5
Types of Group Functions 6-6
Group Functions: Syntax 6-7
Using the AVG and SUM Functions 6-8
Using the MIN and MAX Functions 6-9
Using the COUNT Function 6-10
Using the DISTINCT Keyword 6-11
Group Functions and Null Values in Oracle 6-12
Group Functions and Null Values in MySQL 6-13
Lesson Agenda 6-14
Creating Groups of Data 6-15
Creating Groups of Data: GROUP BY Clause Syntax 6-16
Using the GROUP BY Clause 6-17
Grouping by More Than One Column 6-19
Using the GROUP BY Clause on Multiple Columns 6-20
Illegal Queries Using Group Functions 6-21

Oracle Internal & Oracle Academy Use Only

Illegal Queries Using Group Functions in a WHERE Clause 6-22
Restricting Group Results 6-23
Restricting Group Results with the HAVING Clause 6-24
Using the HAVING Clause 6-25
Lesson Agenda 6-27
Nesting Group Functions in Oracle 6-28
Summary 6-29
Practice 6: Overview 6-30

7 Displaying Data from Multiple Tables Using Joins
Course Roadmap 7-2
Objectives 7-3
Lesson Agenda 7-4
Why Join? 7-5
Obtaining Data from Multiple Tables 7-6
Types of Joins 7-7
Joining Tables Using SQL Syntax 7-8
Lesson Agenda 7-9
Creating Natural Joins 7-10
Retrieving Records with Natural Joins 7-11
Creating Joins with the USING Clause 7-12
Joining Column Names 7-13
Retrieving Records with the USING Clause 7-14
Qualifying Ambiguous Column Names 7-15
Using Table Aliases with the USING Clause in Oracle 7-16
Creating Joins with the ON Clause 7-17
Retrieving Records with the ON Clause 7-18
Creating Three-Way Joins 7-19
Applying Additional Conditions to a Join 7-20
Lesson Agenda 7-21
Joining a Table to Itself 7-22
Self-Joins Using the ON Clause 7-23
Lesson Agenda 7-24
Nonequijoins 7-25
Retrieving Records with Nonequijoins 7-26
Lesson Agenda 7-27
Returning Records with No Direct Match Using OUTER Joins 7-28
INNER Versus OUTER Joins 7-29
LEFT OUTER JOIN 7-30
RIGHT OUTER JOIN 7-31
FULL OUTER JOIN in Oracle 7-32

Oracle Internal & Oracle Academy Use Only

Lesson Agenda 7-33
Cartesian Products 7-34
Generating a Cartesian Product 7-35
Creating Cross Joins 7-36
Summary 7-37
Practice 7: Overview 7-38

8 Using Subqueries to Solve Queries
Course Roadmap 8-2
Objectives 8-3
Lesson Agenda 8-4
Using a Subquery to Solve a Problem 8-5
Subquery Syntax 8-6
Using a Subquery 8-7
Rules and Guidelines for Using Subqueries 8-8
Types of Subqueries 8-9
Lesson Agenda 8-10
Single-Row Subqueries 8-11
Executing Single-Row Subqueries 8-12
Using Group Functions in a Subquery 8-13
HAVING Clause with Subqueries 8-14
What Is Wrong with This Statement? 8-15
No Rows Returned by the Inner Query 8-16
Lesson Agenda 8-17
Multiple-Row Subqueries 8-18
Using the ANY Operator in Multiple-Row Subqueries 8-19
Using the ALL Operator in Multiple-Row Subqueries 8-20
Multiple-Column Subqueries 8-21
Multiple-Column Subquery: Example 8-22
Lesson Agenda 8-23
Null Values in a Subquery 8-24
Summary 8-25
Practice 8: Overview 8-26

9 Using Set Operators
Course Roadmap 9-2
Objectives 9-3
Lesson Agenda 9-4
Set Operators 9-5
Set Operator Rules 9-6
Oracle Server and Set Operators 9-7

Oracle Internal & Oracle Academy Use Only

Lesson Agenda 9-8
Tables Used in This Lesson 9-9
Lesson Agenda 9-13
UNION Operator 9-14
Using the UNION Operator 9-15
UNION ALL Operator 9-16
Using the UNION ALL Operator 9-17
Lesson Agenda 9-18
INTERSECT Operator 9-19
Using the INTERSECT Operator 9-20
Lesson Agenda 9-21
MINUS Operator 9-22
Using the MINUS Operator 9-23
Lesson Agenda 9-24
Matching SELECT Statements in Oracle 9-25
Matching the SELECT Statement: Example in Oracle 9-26
Matching SELECT Statements in MySQL 9-27
Matching the SELECT Statement: Example in MySQL 9-28
Lesson Agenda 9-29
Using the ORDER BY Clause in Set Operations in Oracle 9-30
Using the ORDER BY Clause in Set Operations in Oracle: Example 9-31
Using the ORDER BY Clause with UNION in MySQL 9-32
Using the ORDER BY Clause with UNION: Example in MySQL 9-33
Summary 9-34
Practice 9: Overview 9-35

10A Managing Tables Using DML Statements in Oracle

Course Roadmap 10A-2
Objectives 10A-3
HR Application Scenario 10A-4
Lesson Agenda 10A-5
Data Manipulation Language 10A-6
Adding a New Row to a Table 10A-7
INSERT Statement Syntax 10A-8
Inserting New Rows 10A-9
Inserting Rows with Null Values 10A-10
Inserting Special Values 10A-11
Inserting Specific Date and Time Values 10A-12
Creating a Script 10A-13
Copying Rows from Another Table 10A-14
Lesson Agenda 10A-15

Oracle Internal & Oracle Academy Use Only

Changing Data in a Table 10A-16
UPDATE Statement Syntax 10A-17
Updating Rows in a Table 10A-18
Updating Two Columns with a Subquery 10A-19
Updating Rows Based on Another Table 10A-20
Lesson Agenda 10A-21
Removing a Row from a Table 10A-22
DELETE Statement 10A-23
Deleting Rows from a Table 10A-24
Deleting Rows Based on Another Table 10A-25
TRUNCATE Statement 10A-26
Lesson Agenda 10A-27
Database Transactions 10A-28
Database Transactions: Start and End 10A-29
Advantages of the COMMIT and ROLLBACK Statements 10A-30
Explicit Transaction Control Statements 10A-31
Rolling Back Changes to a Marker 10A-32
Implicit Transaction Processing 10A-33
State of Data Before COMMIT or ROLLBACK 10A-34
State of Data After COMMIT 10A-35
Committing Data 10A-36
State of Data After ROLLBACK 10A-37
State of Data After ROLLBACK: Example 10A-38
Statement-Level Rollback 10A-39
Lesson Agenda 10A-40
Read Consistency 10A-41
Implementing Read Consistency 10A-42
Lesson Agenda 10A-43
FOR UPDATE Clause in a SELECT Statement 10A-44
FOR UPDATE Clause: Examples 10A-45
LOCK TABLE Statement 10A-46
Summary 10A-47
Practice 10a: Overview 10A-48

10B Managing Tables Using DML Statements in MySQL

Course Roadmap 10B-2
Objectives 10B-3
HR Application Scenario 10B-4
Lesson Agenda 10B-5
Data Manipulation Language 10B-6
Adding a New Row to a Table 10B-7

Oracle Internal & Oracle Academy Use Only

INSERT Statement Syntax 10B-8
Inserting New Rows: Listing Column Names 10B-9
Inserting New Rows: Omitting Column Names 10B-10
Inserting Rows with Null Values 10B-11
Inserting Special Values in MySQL 10B-12
Inserting Specific Date and Time Values in MySQL 10B-13
Inserting and Reformatting Specific Date and Time Values in MySQL 10B-14
Copying Rows from Another Table 10B-15
Lesson Agenda 10B-16
Changing Data in a Table 10B-17
UPDATE Statement Syntax 10B-18
Updating Rows in a Table 10B-19
Updating Rows Based on Another Table 10B-20
Quiz 10B-21
Lesson Agenda 10B-22
Removing a Row from a Table 10B-23
DELETE Statement 10B-24
Deleting Rows from a Table 10B-25
Deleting Rows Based on Another Table 10B-26
TRUNCATE Statement 10B-27
Lesson Agenda 10B-28
Multiple-Statement Transactions 10B-29
Transaction Diagram 10B-30
AUTOCOMMIT and Transaction Control Statements 10B-31
Committing Data in a Transaction 10B-32
Rolling Back Changes 10B-33
Rolling Back Changes to a Marker 10B-34
Lesson Agenda 10B-35
Consistent Reads 10B-36
Lesson Agenda 10B-37
FOR UPDATE Clause in a SELECT Statement 10B-38
FOR UPDATE Clause: Examples 10B-39
Summary 10B-40

11A Introduction to Data Definition Language in Oracle

Course Roadmap 11A-2
Objectives 11A-3
HR Application Scenario 11A-4
Lesson Agenda 11A-5
Database Objects 11A-6

Oracle Internal & Oracle Academy Use Only

Naming Rules for Tables and Columns 11A-7
Lesson Agenda 11A-8
CREATE TABLE Statement 11A-9
Creating Tables 11A-10
Lesson Agenda 11A-11
Data Types 11A-12
Datetime Data Types 11A-14
DEFAULT Option 11A-15
Lesson Agenda 11A-16
Including Constraints 11A-17
Constraint Guidelines 11A-18
Defining Constraints 11A-19
Defining Constraints: Example 11A-20
NOT NULL Constraint 11A-21
UNIQUE Constraint 11A-22
PRIMARY KEY Constraint 11A-24
FOREIGN KEY Constraint 11A-25
FOREIGN KEY Constraint: Keywords 11A-27
CHECK Constraint 11A-28
CREATE TABLE: Example 11A-29
Violating Constraints 11A-30
Lesson Agenda 11A-32
Creating a Table Using a Subquery 11A-33
Lesson Agenda 11A-35
ALTER TABLE Statement 11A-36
Adding a Column 11A-38
Modifying a Column 11A-39
Dropping a Column 11A-40
SET UNUSED Option 11A-41
Read-Only Tables 11A-43
Lesson Agenda 11A-44
Dropping a Table 11A-45
Summary 11A-46
Practice 11a: Overview 11A-47

11B Introduction to Data Definition Language in MySQL

Course Roadmap 11B-2
Objectives 11B-3
HR Application Scenario 11B-4
Lesson Agenda 11B-5
Creating a Database: Syntax 11B-6

Oracle Internal & Oracle Academy Use Only

MySQL Naming Conventions 11B-7
Lesson Agenda 11B-8
CREATE TABLE Statement 11B-9
Lesson Agenda 11B-10
Data Types: Overview 11B-11
Numeric Data Types 11B-12
Date and Time Data Types 11B-13
String Data Types 11B-14
Lesson Agenda 11B-15
Indexes, Keys, and Constraints 11B-16
Table Indexes 11B-17
Primary Keys 11B-18
Unique Key Constraints 11B-19
Foreign Key Constraints 11B-20
Foreign Key Constraint: Example Tables 11B-21
FOREIGN KEY Constraint: Example Statement 11B-22
FOREIGN KEY Constraint: Referential Actions 11B-23
Secondary Indexes 11B-24
Lesson Agenda 11B-25
Column Options 11B-26
Lesson Agenda 11B-27
Creating a Table Using a Subquery 11B-28
Creating a Table Using a Subquery: Example 11B-29
Lesson Agenda 11B-30
ALTER TABLE Statement 11B-31
ALTER TABLE Statement: Add, Modify, or Drop Columns 11B-32
Adding a Column 11B-33
Modifying a Column 11B-34
Dropping a Column 11B-35
ALTER TABLE Statement: Add an Index or Constraint 11B-36
ALTER TABLE to Add a Constraint or Index: Example 11B-37
Creating Indexes by Using the CREATE INDEX Statement 11B-38
Viewing Index Definitions by Using the SHOW INDEX Statement 11B-39
Showing How a Table Was Created with the SHOW CREATE TABLE Statement 11B-40
Lesson Agenda 11B-41
Dropping a Table 11B-42
Summary 11B-43

Oracle Internal & Oracle Academy Use Only

12 Introduction to Data Dictionary Views
Course Roadmap 12-2
Objectives 12-3
Lesson Agenda 12-4
Why Data Dictionary? 12-5
Data Dictionary 12-6
Data Dictionary Structure 12-7
How to Use Dictionary Views 12-9
USER_OBJECTS and ALL_OBJECTS Views 12-10
USER_OBJECTS View 12-11
Lesson Agenda 12-12
Table Information 12-13
Column Information 12-14
Constraint Information 12-16
USER_CONSTRAINTS: Example 12-17
Querying USER_CONS_COLUMNS 12-18
Lesson Agenda 12-19
Adding Comments to a Table 12-20
Summary 12-21
Practice 12: Overview 12-22

13 Creating Sequences, Synonyms, and Indexes
Course Roadmap 13-2
Objectives 13-3
Lesson Agenda 13-4
E-Commerce Scenario 13-5
Database Objects 13-6
Referencing Another User's Tables 13-7
Sequences 13-8
CREATE SEQUENCE Statement: Syntax 13-9
Creating a Sequence 13-11
NEXTVAL and CURRVAL Pseudocolumns 13-12
Using a Sequence 13-14
SQL Column Defaulting Using a Sequence 13-15
Caching Sequence Values 13-16
Modifying a Sequence 13-17
Guidelines for Modifying a Sequence 13-18
Sequence Information 13-19
Lesson Agenda 13-20
Synonyms 13-21
Creating a Synonym for an Object 13-22

Oracle Internal & Oracle Academy Use Only

Creating and Removing Synonyms 13-23
Synonym Information 13-24
Lesson Agenda 13-25
Indexes 13-26
How Are Indexes Created? 13-27
Creating an Index 13-28
CREATE INDEX with the CREATE TABLE Statement 13-29
Function-Based Indexes 13-31
Creating Multiple Indexes on the Same Set of Columns 13-32
Creating Multiple Indexes on the Same Set of Columns: Example 13-33
Index Information 13-34
USER_INDEXES: Examples 13-35
Querying USER_IND_COLUMNS 13-36
Removing an Index 13-37
Summary 13-38
Practice 13: Overview 13-39

14 Creating Views
Course Roadmap 14-2
Objectives 14-3
Lesson Agenda 14-4
Why Views? 14-5
Database Objects 14-6
What Is a View? 14-7
Advantages of Views 14-8
Simple Views and Complex Views 14-9
Lesson Agenda 14-10
Creating a View 14-11
Retrieving Data from a View 14-14
Modifying a View 14-15
Creating a Complex View 14-16
View Information 14-17
Lesson Agenda 14-18
Rules for Performing DML Operations on a View 14-19
Rules for Performing Modify Operations on a View 14-20
Rules for Performing Insert Operations Through a View 14-21
Using the WITH CHECK OPTION Clause 14-22
Denying DML Operations 14-23
Lesson Agenda 14-25

Oracle Internal & Oracle Academy Use Only

Removing a View 14-26
Summary 14-27
Practice 14: Overview 14-28

15 Managing Schema Objects
Course Roadmap 15-2
Objectives 15-3
Lesson Agenda 15-4
Adding a Constraint Syntax 15-5
Adding a Constraint 15-6
Dropping a Constraint 15-7
Dropping a Constraint ONLINE 15-8
ON DELETE Clause 15-9
Cascading Constraints 15-10
Renaming Table Columns and Constraints 15-12
Disabling Constraints 15-13
Enabling Constraints 15-14
Constraint States 15-15
Deferring Constraints 15-16
Difference Between INITIALLY DEFERRED and INITIALLY IMMEDIATE 15-17
DROP TABLE … PURGE 15-19
Lesson Agenda 15-20
Using Temporary Tables 15-21
Temporary Table 15-22
Temporary Table Characteristics 15-23
Creating a Global Temporary Table 15-24
Creating a Private Temporary Table 15-25
Lesson Agenda 15-26
External Tables 15-27
Creating a Directory for the External Table 15-28
Creating an External Table 15-30
Creating an External Table by Using ORACLE_LOADER 15-32
Querying External Tables 15-33
Creating an External Table by Using ORACLE_DATAPUMP: Example 15-34
Summary 15-35
Practice 15: Overview 15-36

16 Retrieving Data by Using Subqueries
Course Roadmap 16-2
Objectives 16-3
Lesson Agenda 16-4

Oracle Internal & Oracle Academy Use Only

Retrieving Data by Using a Subquery as a Source 16-5
Lesson Agenda 16-7
Multiple-Column Subqueries 16-8
Column Comparisons 16-9
Pairwise Comparison Subquery 16-10
Nonpairwise Comparison Subquery 16-11
Lesson Agenda 16-12
Scalar Subquery Expressions 16-13
Scalar Subqueries: Examples 16-14
Lesson Agenda 16-15
Correlated Subqueries 16-16
Using Correlated Subqueries: Example 1 16-18
Using Correlated Subqueries: Example 2 16-19
Lesson Agenda 16-20
Using the EXISTS Operator 16-21
Find All Departments That Do Not Have Any Employees 16-23
Lesson Agenda 16-24
WITH Clause 16-25
WITH Clause: Example 16-26
Recursive WITH Clause 16-27
Recursive WITH Clause: Example 16-28
Summary 16-29
Practice 16: Overview 16-30

17 Manipulating Data by Using Subqueries
Course Roadmap 17-2
Objectives 17-3
Lesson Agenda 17-4
Using Subqueries to Manipulate Data 17-5
Lesson Agenda 17-6
Inserting by Using a Subquery as a Target 17-7
Lesson Agenda 17-9
Using the WITH CHECK OPTION Keyword on DML Statements 17-10
Lesson Agenda 17-12
Correlated UPDATE 17-13
Using Correlated UPDATE 17-14
Correlated DELETE 17-16
Using Correlated DELETE 17-17
Summary 17-18
Practice 17: Overview 17-19

Oracle Internal & Oracle Academy Use Only

18 Controlling User Access
Course Roadmap 18-2
Objectives 18-3
Lesson Agenda 18-4
Controlling User Access 18-5
Privileges 18-6
System Privileges 18-7
Creating Users 18-8
User System Privileges 18-9
Granting System Privileges 18-10
Lesson Agenda 18-11
What Is a Role? 18-12
Creating and Granting Privileges to a Role 18-13
Changing Your Password 18-14
Lesson Agenda 18-15
Object Privileges 18-16
Granting Object Privileges 18-18
Passing On Your Privileges 18-19
Confirming Granted Privileges 18-20
Lesson Agenda 18-21
Revoking Object Privileges 18-22
Summary 18-24
Practice 18: Overview 18-25

19 Manipulating Data Using Advanced Queries
Course Roadmap 19-2
Objectives 19-3
Lesson Agenda 19-4
Explicit Default Feature: Overview 19-5
Using Explicit Default Values 19-6
Lesson Agenda 19-7
E-Commerce Scenario 19-8
Multitable INSERT Statements: Overview 19-9
Types of Multitable INSERT Statements 19-11
Multitable INSERT Statements 19-12
Unconditional INSERT ALL 19-14
Conditional INSERT ALL: Example 19-15
Conditional INSERT ALL 19-16
Conditional INSERT FIRST: Example 19-18
Conditional INSERT FIRST 19-19
Pivoting INSERT 19-20

Oracle Internal & Oracle Academy Use Only

Lesson Agenda 19-23
MERGE Statement 19-24
MERGE Statement Syntax 19-25
Merging Rows: Example 19-26
Lesson Agenda 19-29
FLASHBACK TABLE Statement 19-30
Using the FLASHBACK TABLE Statement 19-32
Lesson Agenda 19-33
Tracking Changes in Data 19-34
Flashback Query: Example 19-35
Flashback Version Query: Example 19-36
VERSIONS BETWEEN Clause 19-37
Summary 19-38
Practice 19: Overview 19-39

20 Managing Data in Different Time Zones
Course Roadmap 20-2
Objectives 20-3
Lesson Agenda 20-4
E-Commerce Scenario 20-5
Time Zones 20-6
TIME_ZONE Session Parameter 20-7
CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP 20-8
Comparing Date and Time in a Session's Time Zone 20-9
DBTIMEZONE and SESSIONTIMEZONE 20-11
TIMESTAMP Data Types 20-12
TIMESTAMP Fields 20-13
Difference Between DATE and TIMESTAMP 20-14
Comparing TIMESTAMP Data Types 20-15
Lesson Agenda 20-16
INTERVAL Data Types 20-17
INTERVAL Fields 20-18
INTERVAL YEAR TO MONTH: Example 20-19
INTERVAL DAY TO SECOND Data Type: Example 20-21
Lesson Agenda 20-22
EXTRACT 20-23
TZ_OFFSET 20-24
FROM_TZ 20-26
TO_TIMESTAMP 20-27
TO_YMINTERVAL 20-28
TO_DSINTERVAL 20-29

Oracle Internal & Oracle Academy Use Only

Daylight Saving Time (DST) 20-30
Summary 20-32
Practice 20: Overview 20-33

21 Conclusion
Course Goals 21-2
Oracle University: Oracle SQL Training 21-3
Oracle University: MySQL Training 21-4
Oracle SQL References 21-5
MySQL Websites 21-6
Your Evaluation 21-7
Thank You 21-8
Q&A Session 21-9

A Table Descriptions

B Using SQL Developer
Objectives B-2
What Is Oracle SQL Developer? B-3
Specifications of SQL Developer B-4
SQL Developer 17.4.1 Interface B-5
Creating a Database Connection B-7
Browsing Database Objects B-10
Displaying the Table Structure B-11
Browsing Files B-12
Creating a Schema Object B-13
Creating a New Table: Example B-14
Using the SQL Worksheet B-15
Executing SQL Statements B-18
Saving SQL Scripts B-19
Executing Saved Script Files: Method 1 B-20
Executing Saved Script Files: Method 2 B-21
Formatting the SQL Code B-22
Using Snippets B-23
Using Snippets: Example B-24
Using the Recycle Bin B-25
Debugging Procedures and Functions B-26
Database Reporting B-27
Creating a User-Defined Report B-28
External Tools B-29
Setting Preferences B-30

Oracle Internal & Oracle Academy Use Only

Data Modeler in SQL Developer B-31
Summary B-32

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

D Commonly Used SQL Commands
Objectives D-2
Basic SELECT Statement D-3
SELECT Statement D-4
WHERE Clause D-5
ORDER BY Clause D-6
GROUP BY Clause D-7
Data Definition Language D-8
CREATE TABLE Statement D-9
ALTER TABLE Statement D-10
DROP TABLE Statement D-11
GRANT Statement D-12
Privilege Types D-13
REVOKE Statement D-14
TRUNCATE TABLE Statement D-15
Data Manipulation Language D-16
INSERT Statement D-17
UPDATE Statement Syntax D-18
DELETE Statement D-19
Transaction Control Statements D-20
COMMIT Statement D-21

Oracle Internal & Oracle Academy Use Only

ROLLBACK Statement D-22
SAVEPOINT Statement D-23
Joins D-24
Types of Joins D-25
Qualifying Ambiguous Column Names D-26
Natural Join D-27
Equijoins D-28
Retrieving Records with Equijoins D-29
Additional Search Conditions Using the AND No equijoin WHERE Operators D-30
Retrieving Records with Nonequijoins D-31
Retrieving Records by Using the USING Clause D-32
Retrieving Records by Using the ON Clause D-33
Left Outer Join D-34
Right Outer Join D-35
Full Outer Join D-36
Self-Join: Example D-37
Cross Join D-38
Summary D-39

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

Oracle Internal & Oracle Academy Use Only

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

G Writing Advanced Scripts
Objectives G-2
Using SQL to Generate SQL G-3
Creating a Basic Script G-4
Controlling the Environment G-5
The Complete Picture G-6
Dumping the Contents of a Table to a File G-7
Generating a Dynamic Predicate G-9
Summary G-11

H Oracle Database Architectural Components
Objectives H-2
Oracle Database Architecture: Overview H-3
Oracle Database Server Structures H-4
Connecting to the Database H-5
Interacting with an Oracle Database H-6
Oracle Memory Architecture H-8
Process Architecture H-10
Database Writer Process H-12
Log Writer Process H-13
Checkpoint Process H-14
System Monitor Process H-15
Process Monitor Process H-16
Oracle Database Storage Architecture H-17
Logical and Physical Database Structures H-19
Processing a SQL Statement H-21
Processing a Query H-22
Shared Pool H-23

Oracle Internal & Oracle Academy Use Only

Database Buffer Cache H-25
Program Global Area H-26
Processing a DML Statement H-27
Redo Log Buffer H-29
Rollback Segment H-30
COMMIT Processing H-31
Summary of the Oracle Database Architecture H-33
Summary H-34

I Regular Expression Support
Objectives I-2
What Are Regular Expressions? I-3
Benefits of Using Regular Expressions I-4
Using the Regular Expressions Functions and Conditions in SQL and PL/SQL I-5
What Are Metacharacters? I-6
Using Metacharacters with Regular Expressions I-7
Regular Expressions Functions and Conditions: Syntax I-9
Performing a Basic Search by Using the REGEXP_LIKE Condition I-10
Replacing Patterns by Using the REGEXP_REPLACE Function I-11
Finding Patterns by Using the REGEXP_INSTR Function I-12
Extracting Substrings by Using the REGEXP_SUBSTR Function I-13
Subexpressions I-14
Using Subexpressions with Regular Expression Support I-15
Why Access the nth Subexpression? I-16
REGEXP_SUBSTR: Example I-17
Using the REGEXP_COUNT Function I-18
Regular Expressions and Check Constraints: Examples I-19
Quiz I-20
Summary I-21

Oracle Internal & Oracle Academy Use Only

ORACLE

1

Introduction

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Define the goals of the course
- List the features of Oracle Database 19c
- 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)
- Identify the development environments that can be used for this course
- Describe the database and schema used in this course

2

In this lesson, you gain an understanding of RDBMS. You are also introduced to Oracle SQL Developer and SQL*Plus as development environments used for executing SQL statements, and for formatting and reporting purposes.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Course objectives, roadmap, and appendixes used in the course
- Overview of Oracle Database 19c 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
- Oracle Database 19c SQL Documentation and Additional Resources

3

Oracle Internal & Oracle Academy Use Only

Course Objectives

After completing this course, you should be able to:

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

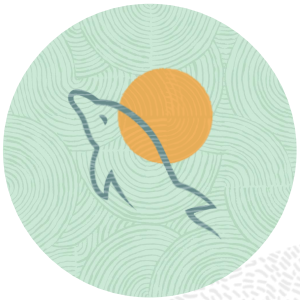
4

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.

Oracle Internal & Oracle Academy Use Only

Icons Used in This Course

Indicates the output from Oracle Database



Indicates the output from MySQL

5

The above icons have been used throughout the course to indicate the output results from Oracle Database and MySQL.

The respective icons have also been used in the title of the slides to indicate exclusive Oracle Database or MySQL only content.

Look out for these icons throughout the course!

Oracle Internal & Oracle Academy Use Only

Course Roadmap

- Lesson 1: Introduction
- **Unit 1: Retrieving, Restricting, and Sorting Data**
  - Lesson 2: Retrieving Data Using the SQL `SELECT` Statement
  - Lesson 3: Restricting and Sorting Data
  - Lesson 4: Using Single-Row Functions to Customize Output
  - Lesson 5: Using Conversion Functions and Conditional Expressions
- Unit 2: Joins, Subqueries, and Set Operators
- Unit 3: DML and DDL

6

In Unit 1, you will learn how to query data from tables, how to query selected records from tables, and also how to sort the data retrieved from the tables.

Oracle Internal & Oracle Academy Use Only

Course Roadmap

- Lesson 1: Introduction
- Unit 1: Retrieving, Restricting, and Sorting Data
- **Unit 2: Joins, Subqueries, and Set Operators**
- Unit 3: DML and DDL

- Lesson 6: Reporting Aggregated Data Using the Group Functions
- Lesson 7: Displaying Data from Multiple Tables Using Joins
- Lesson 8: Using Subqueries to Solve Queries
- Lesson 9: Using Set Operators

7

In Unit 2, you will learn about SQL statements to query and display data from multiple tables using Joins. You will also learn to use subqueries when the condition is unknown, use group functions to aggregate data, and use set operators.

Oracle Internal & Oracle Academy Use Only

Course Roadmap

- Lesson 1: Introduction
- Unit 1: Retrieving, Restricting, and Sorting Data
- Unit 2: Joins, Subqueries, and Set Operators
- **Unit 3: DML and DDL**
  - Lesson 10: Managing Tables Using DML Statements
  - Lesson 11: Introduction to Data Definition Language

8

In Unit 3, you will learn how to create and manage database objects using DDL statements. You will also learn how to manage data in the tables using DML statements.

Oracle Internal & Oracle Academy Use Only

Course Roadmap

Unit 4: Views, Sequences, Synonyms, and Indexes

- Lesson 12: Introduction to Data Dictionary Views
- Lesson 13: Creating Sequences, Synonyms, and Indexes
- Lesson 14: Creating Views

Unit 5: Managing Database Objects and Subqueries

Unit 6: User Access

Unit 7: Advanced Queries

9

In Unit 4, you get an introduction to data dictionary views. You learn how to create views on tables. You also learn about synonyms, sequences, and indexes, and how to create them.

Oracle Internal & Oracle Academy Use Only

Course Roadmap

- Unit 4: Views, Sequences, Synonyms, and Indexes
- **Unit 5: Managing Database Objects and Subqueries**
  - Lesson 15: Managing Schema Objects
  - Lesson 16: Retrieving Data by Using Subqueries
  - Lesson 17: Manipulating Data by Using Subqueries
- Unit 6: User Access
- Unit 7: Advanced Queries

10

In Unit 4, you get an introduction to data dictionary views. You learn how to create views on tables. You also learn about synonyms, sequences, and indexes, and how to create them.

Oracle Internal & Oracle Academy Use Only

Course Roadmap

Unit 4: Views, Sequences, Synonyms, and Indexes

Unit 5: Managing Database Objects and Subqueries

Unit 6: User Access

Lesson 18: Controlling User Access

Unit 7: Advanced Queries

11

In Unit 7, you will learn to use advanced SQL statements to manage data. You will also learn to manipulate data in different time zones.

Oracle Internal & Oracle Academy Use Only

Course Roadmap

Unit 4: Views, Sequences, Synonyms, and Indexes

Unit 5: Managing Database Objects and Subqueries

Unit 6: User Access

Unit 7: Advanced Queries

- Lesson 19: Manipulating Data Using Advanced Queries
- Lesson 20: Managing Data in Different Time Zones

12

In Unit 7, you will learn to use advanced SQL statements to manage data. You will also learn to manage data in different time zones.

Oracle Internal & Oracle Academy Use Only

Appendixes and Practices Used in the Course

- Activity Guides for Oracle and MySQL
- Appendix A: Table Descriptions
- Appendix B: Using SQL Developer
- Appendix C: Using SQL*Plus
- Appendix D: Commonly Used SQL Commands
- Appendix E: Generating Reports by Grouping Related Data
- Appendix F: Hierarchical Retrieval
- Appendix G: Writing Advanced Scripts
- Appendix H: Oracle Database Architectural Components
- Appendix I: Regular Expression Support
  - Practices and Solutions
  - Additional Practices and Solutions

13

The course has one student guide which covers topics on both Oracle and MySQL.

There are separate activity guides for Oracle and MySQL. The students can choose to complete either of the activity guides depending on the environment he/she chooses.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Course objectives, roadmap, and appendixes used in the course
- Overview of Oracle Database 19c 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
- Oracle Database 19c SQL Documentation and Additional Resources

14

Oracle Internal & Oracle Academy Use Only

Oracle Database 19c: Focus Areas

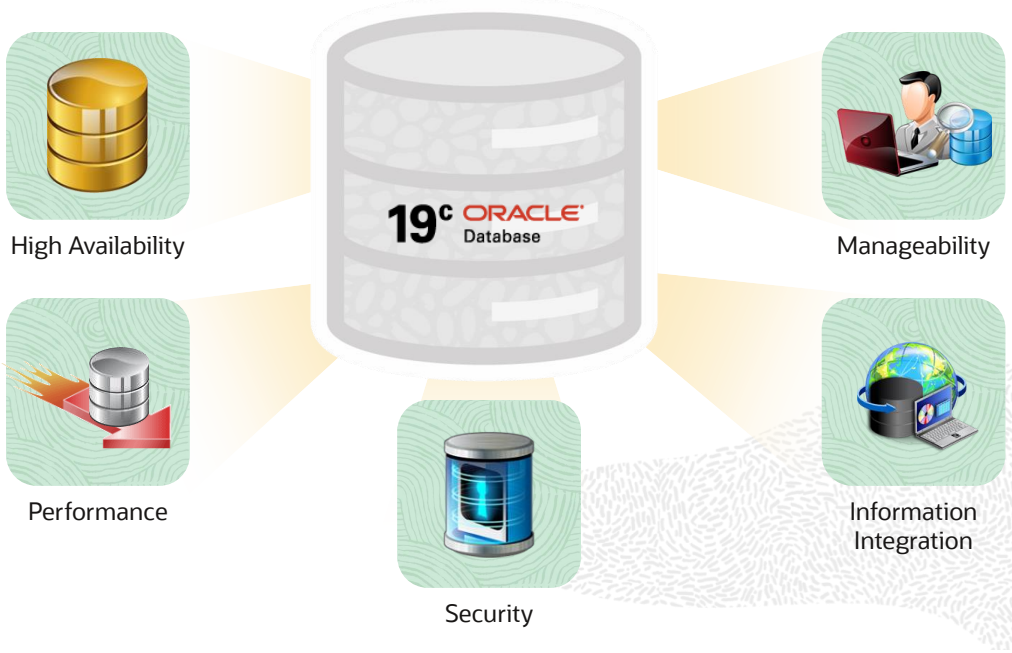
15

By using Oracle Database 19c, 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 19c also helps you to consolidate and extend the benefits of grid computing and manage changes in a controlled and cost-effective manner.
- Oracle Database 19c 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 19c, 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 19c. This will help you to standardize, consolidate, and automate database services on the cloud.

Oracle Internal & Oracle Academy Use Only

Oracle Database 19c



16

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 19c 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 19c 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 19c enables organizations to manage large, scalable, transactional, and data warehousing systems that deliver fast data access using low-cost modular storage.
- **Security:** Oracle Database 19c helps in protecting your information with unique secure configurations, data encryption and masking, and sophisticated auditing capabilities.

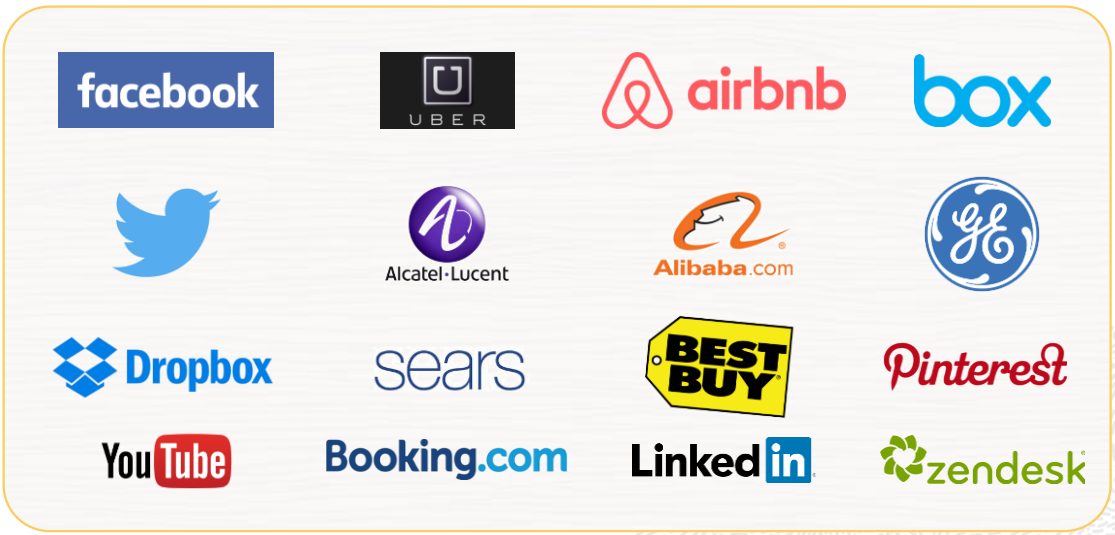
Oracle Internal & Oracle Academy Use Only

- **Information integration:** You can utilize Oracle Database 19c 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 19c's advanced information lifecycle management capabilities.

Oracle Internal & Oracle Academy Use Only

MySQL: A Modern Database for the Digital Age

Digital Disruptors and Large Enterprises Rely on MySQL to Innovate



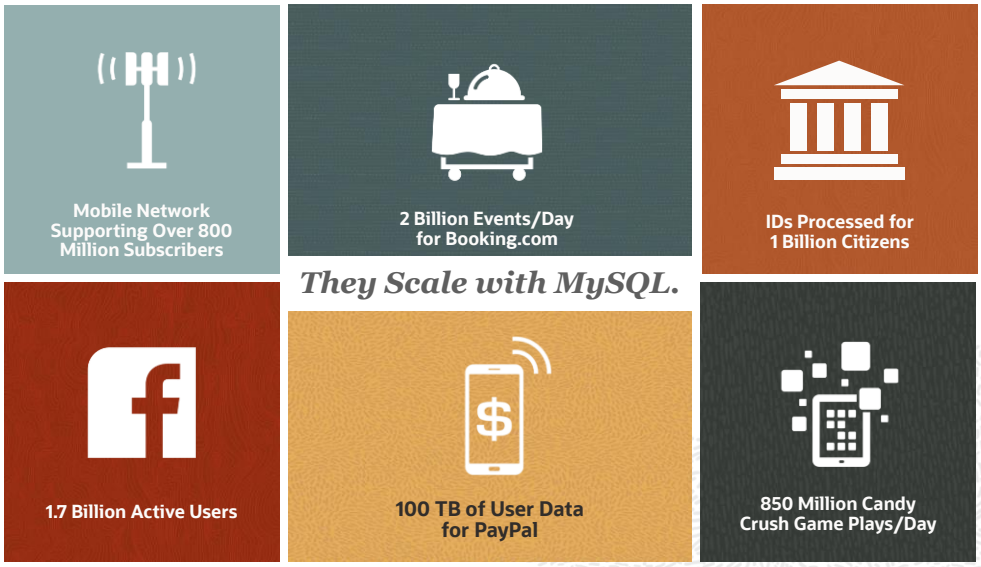
18

The slide lists several digital disruptors and companies that have redefined customer expectations. They rely on MySQL, and so do large enterprises that drive digital transformation initiatives.

They choose MySQL because it is a modern database designed for web-based applications, which allows them to innovate quickly. But those are not the only reasons. As the world's most popular open-source database with over 20 years of existence, MySQL is a proven, battle-tested, and mature technology.

Oracle Internal & Oracle Academy Use Only

High Scalability with MySQL



19

- MySQL Cluster powers a mobile network in Asia that has a count of over 800 million subscribers.
- A government, also in Asia, has processed over one billion IDs for its citizens.

Oracle Internal & Oracle Academy Use Only

MySQL-Supported Operating Systems

MySQL:

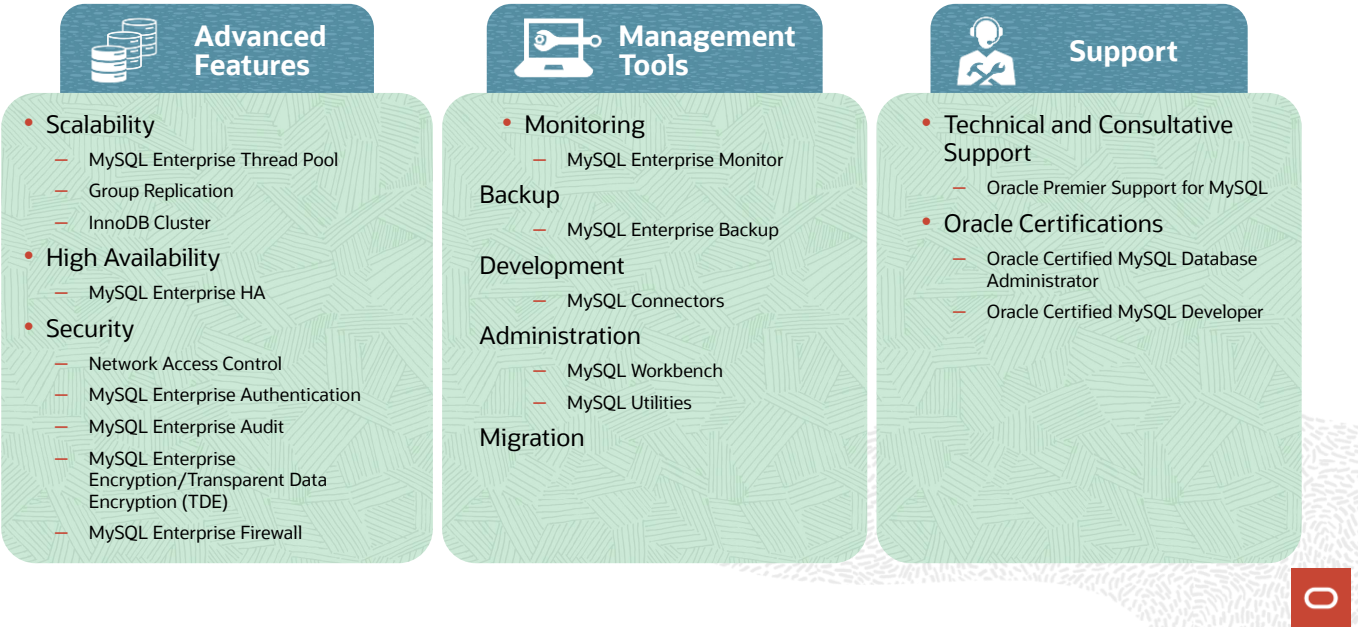
- Provides control and flexibility for users
- Supports multiple commodity platforms, including:
  - Windows (x86, x86_64)
  - Linux (x86, x86_64)
  - Oracle Solaris (SPARC_64, x86_64, x86)
  - Mac OS X (x86, x86_64)
- Can be compiled to run on other platforms

20

For a full list of supported operating systems, see the website at: http://www.mysql.com/support/supportedplatforms/database.html.

Oracle Internal & Oracle Academy Use Only

MySQL Enterprise Edition



Scalability

- MySQL Enterprise Thread Pool allows you to scale the performance of your application in the face of increasing user, query, and data loads.
- The MySQL Group Replication feature is a multi-master, update-anywhere replication plug-in for MySQL.
- InnoDB Cluster is an integrated, native, full stack high availability solution for MySQL.

MySQL Enterprise High Availability

MySQL Enterprise High Availability enables you to meet the availability requirements of even the most demanding, mission-critical applications. MySQL Group Replication provides native high availability with built-in group membership management, data consistency guarantees, conflict detection and handling, node failure detection, and database failover–related operations, all without the need for manual intervention or custom tooling.

MySQL Security

- **Network Access Control:** Restrict connections to your MySQL instances.
- **MySQL Enterprise Authentication:** Authenticate MySQL users against your existing directory services and security rules.
- **MySQL Enterprise Audit:** Generate a complete audit trail to track MySQL access and usage.
- **MySQL Enterprise Encryption:** Protect the sensitive data that is stored in MySQL databases, in backups, or during transfer.
- **MySQL Transparent Data Encryption (TDE):** Protect data at rest and securely manage your encryption keys.
- **MySQL Enterprise Firewall:** Ensure real-time protection against database-specific attacks.

Oracle Internal & Oracle Academy Use Only

Why MySQL Enterprise Edition?

In addition to all the features you love...

- Insure your deployment.
- Get the best results.
- Delight customers.

22

Oracle Internal & Oracle Academy Use Only

Oracle Premier Support for MySQL

- Largest MySQL engineering and support organization
- Backed by MySQL developers
- World-class support in 29 languages
- Hot fixes and maintenance releases
- 24 x 7 x 365
- Unlimited incidents
- Consultative support
- Global scale and reach

Get immediate help for any MySQL issue, plus expert advice.

"The MySQL support service has been essential in helping us with troubleshooting and providing recommendations for the production cluster. Thanks."

– Carlos Morales (playfulplay.com)

23

Oracle Internal & Oracle Academy Use Only

MySQL and Oracle Integration

MySQL Integrates into the Oracle Environment

- Oracle Linux
- Oracle Solaris
- Oracle OpenStack
- Oracle Fusion Middleware
- Oracle Enterprise Manager
- Oracle Secure Backup
- Oracle Clusterware
- Oracle Audit Vault and Database Firewall

MySQL

24

In addition to Oracle Enterprise Manager, MySQL is now integrated with nearly all relevant Oracle products.

Oracle wants to make it very easy for existing Oracle customers to integrate and manage MySQL in their current environment.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c 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
- Oracle Database 19c SQL Documentation and Additional Resources

25

Oracle Internal & Oracle Academy Use Only

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

26

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, gluesticks 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.

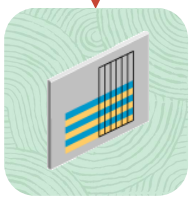
For more information about the relational and object relational model, refer to *Oracle Database Concepts for 19c Database*.

Oracle Internal & Oracle Academy Use Only

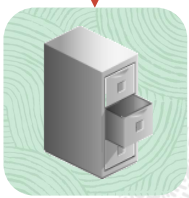
Data Storage on Different Media

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping		
4	60	IT		
5	80	Sales		
6	90	Executive		
7	110	Accounting		
8	190	Contracting		

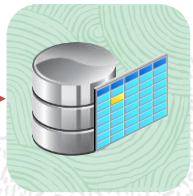
	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000



Electronic spreadsheet



Filing cabinet



Database

27

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

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

A *database* is an organized collection of information.

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

Oracle Internal & Oracle Academy Use Only

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:
  - Collection of objects or relations
  - Set of operators to act on the relations
  - Data integrity for accuracy and consistency

28

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.

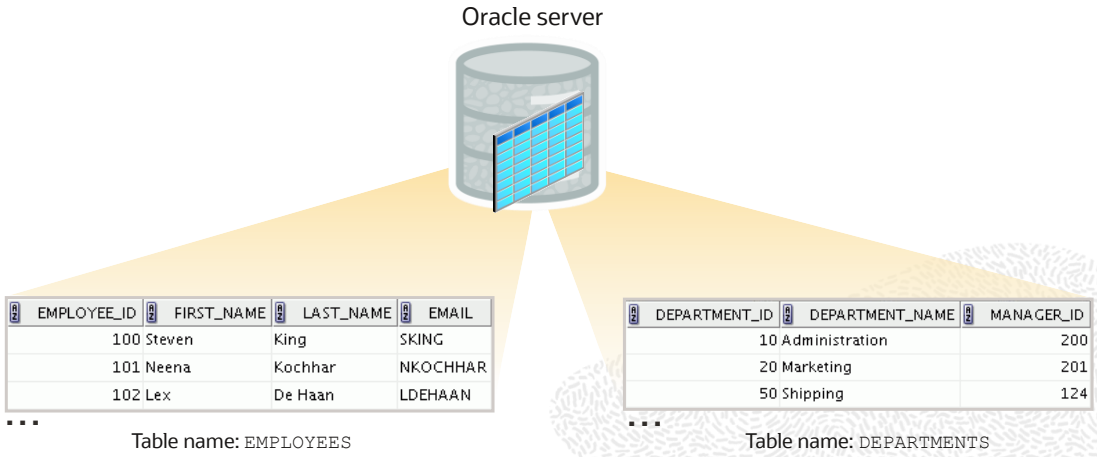
Components of the Relational Model

- 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

Oracle Internal & Oracle Academy Use Only

Definition of a Relational Database

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



29

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

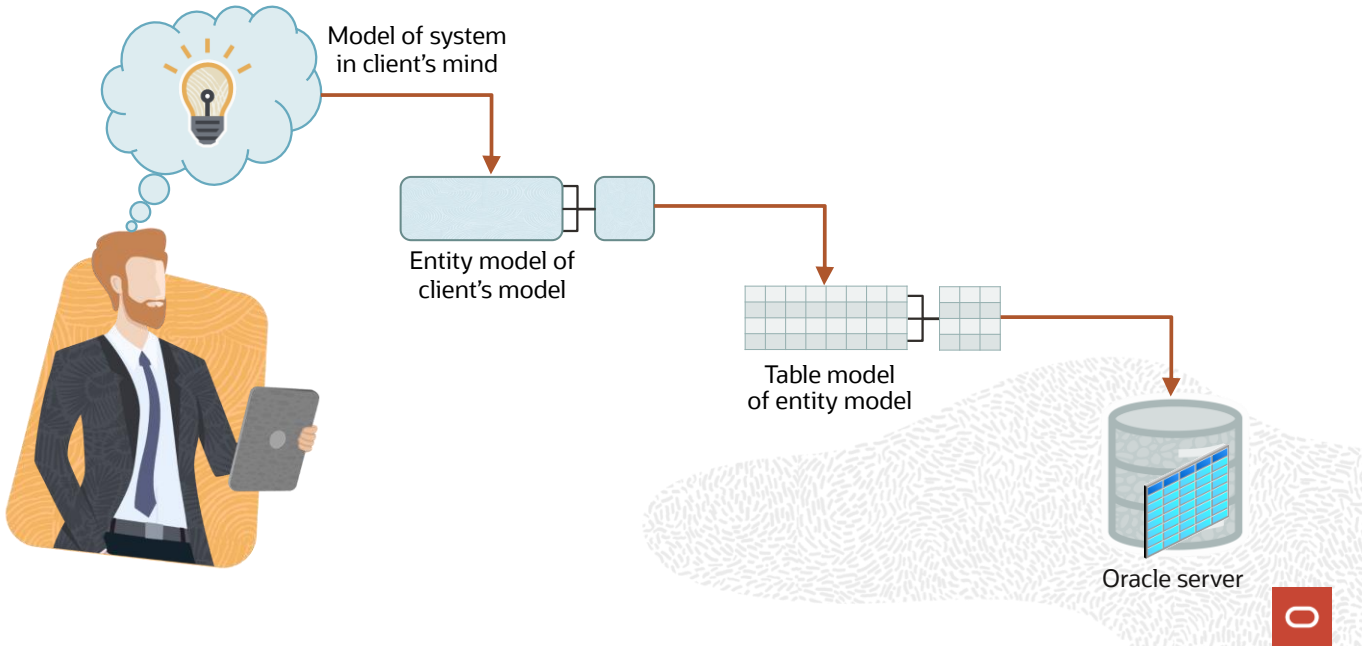
For example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table, and a salary table.

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.

Oracle Internal & Oracle Academy Use Only

Data Models



30

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:

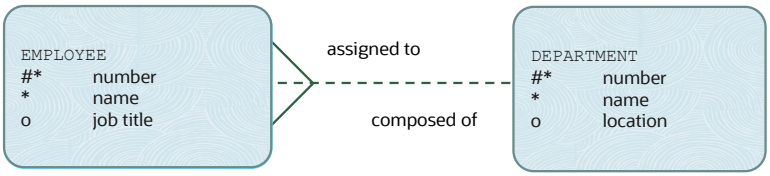
- Communicate
- Categorize
- Describe
- Specify
- Investigate
- Evolve
- Analyze
- Imitate

The objective is to produce a model that fits a multitude of these uses, can be understood by an end user, and contains sufficient details for a developer to build a database system. An Entity-Relationship model is one such data model.

Oracle Internal & Oracle Academy Use Only

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..."

31

Entity relationship model is widely used because in an effective system, data is divided into discrete categories or entities.

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

Benefits of ER Modeling

You can:

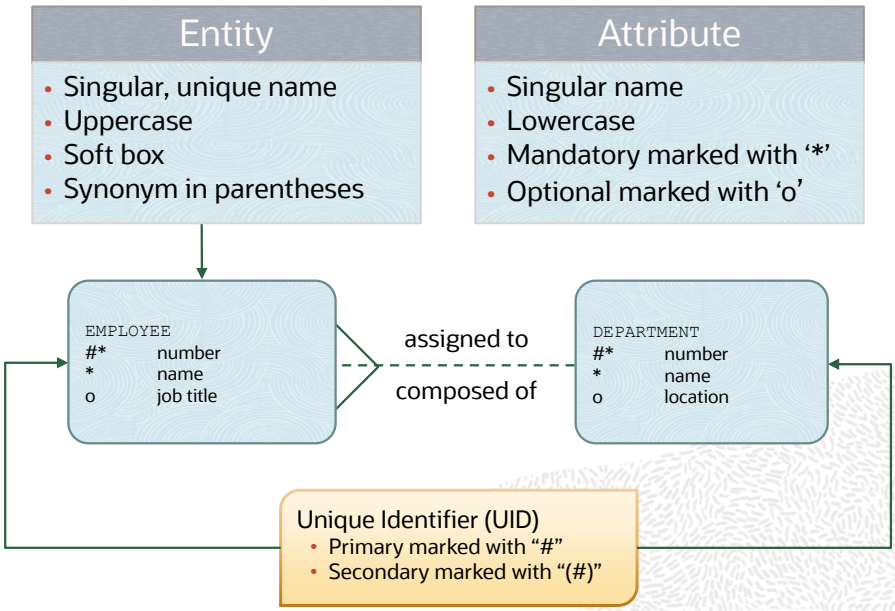
- Document information for the organization in a clear, precise format
- Provide a clear picture of the scope of the information requirement
- Provide an easily understood pictorial map for database design
- Offer an effective framework for integrating multiple applications

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.

Oracle Internal & Oracle Academy Use Only

Entity Relationship Modeling Conventions



32

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)

Oracle Internal & Oracle Academy Use Only

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 a unique identifier in `EMPLOYEE` entity since no two employees can have the same employee ID.

Oracle Internal & Oracle Academy Use Only

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.

Table name: EMPLOYEES

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

...

Primary key

Foreign key

Table name: DEPARTMENTS

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

Primary key

34

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.

Oracle Internal & Oracle Academy Use Only

Relational Database Terminology

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	90
101	Neena	Kochhar	17000	(null)	90
102	Lex	De Haan	17000	(null)	90
103	Alexander	Hunold	9000	(null)	60
104	Bruce	Ernst	6000	(null)	60
107	Diana	Lorentz	4200	(null)	60
124	Kevin	Mourgos	5800	(null)	50
141	Trenna	Rajs	3500	(null)	50
142	Curtis	Davies	3100	(null)	50
143	Randall	Matos	2600	(null)	50
144	Peter	Vargas	2500	(null)	50
149	Eleni	Zlotkey	10500	0.2	80
174	Ellen	Abel	11000	0.3	80
176	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

35

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.

Oracle Internal & Oracle Academy Use Only

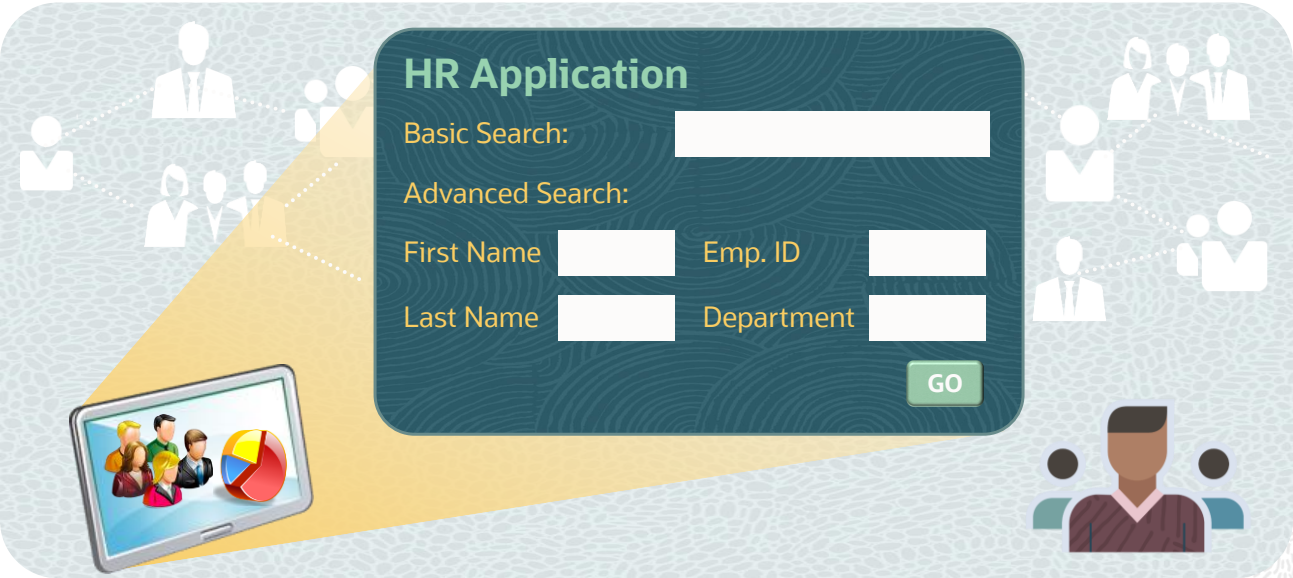
Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c 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
- Oracle Database 19c SQL Documentation and Additional Resources

36

Oracle Internal & Oracle Academy Use Only

Human Resources (HR) Application



37

The slide shows an example of a sample HR application, which is generally used by the HR department in an 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. 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 to delete employee records of employees who have quit. In addition to these, 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, the average salaries in each department, the list of employees who received a bonus this year, etc.

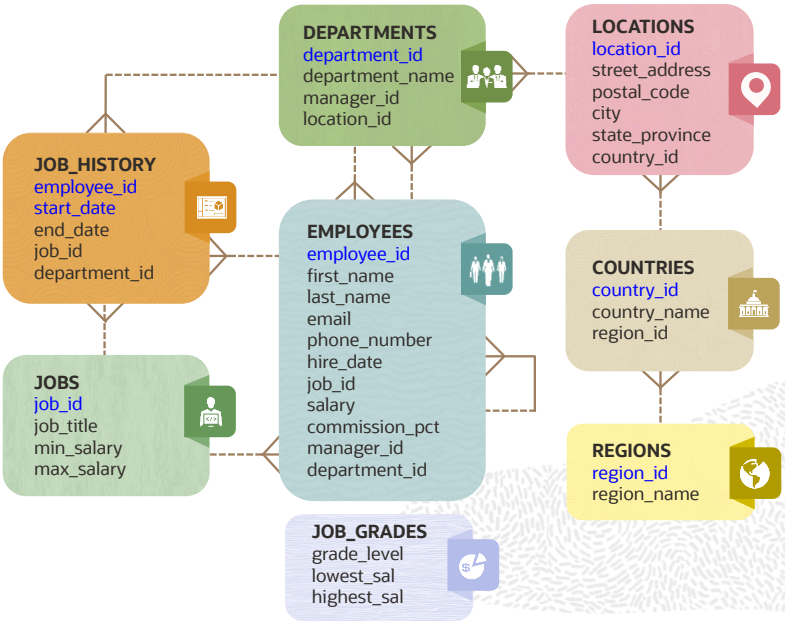
In this course, we will discuss various scenarios based on the sample HR application. You will also see different examples in the lessons based on HR schema.

Note: Screens like the one shown in this slide are a mockup of an application that could be developed using a tool like APEX or in a programming language like Java or PHP. This course will not be concerned about developing those screens but rather about the database maintaining the data for the HR application. Oracle provides courses on application development in APEX, and MySQL provides training in the MySQL for Developers course on application development in Java, PHP, and Python.

For more information, visit *https://education.oracle.com/home*

Oracle Internal & Oracle Academy Use Only

Tables Used in This Course



38

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.

Oracle Internal & Oracle Academy Use Only

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	EZLOTKEY	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	MK_MAN	13000
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_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

39

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c 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
- Oracle Database 19c SQL Documentation and Additional Resources

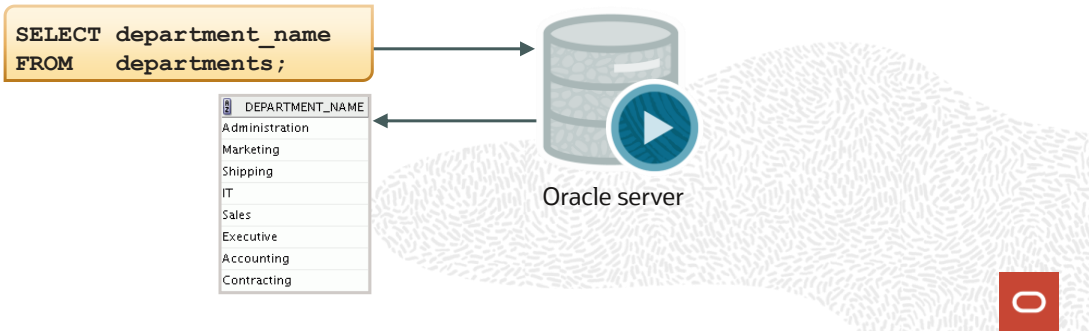
40

Oracle Internal & Oracle Academy Use Only

Using SQL to Query Your Database

Structured query language (SQL) is:

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



41

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 2016). The Oracle Database complies with the ANSI and ISO standards for SQL (mandatory portion compliance is known as Core SQL, which is at SQL:2016 for the 19c version of the Oracle database).

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.

Oracle Internal & Oracle Academy Use Only

How SQL Works

SQL:

- Is standalone and powerful
- Processes groups of data
- Lets you work with data at a logical level

42

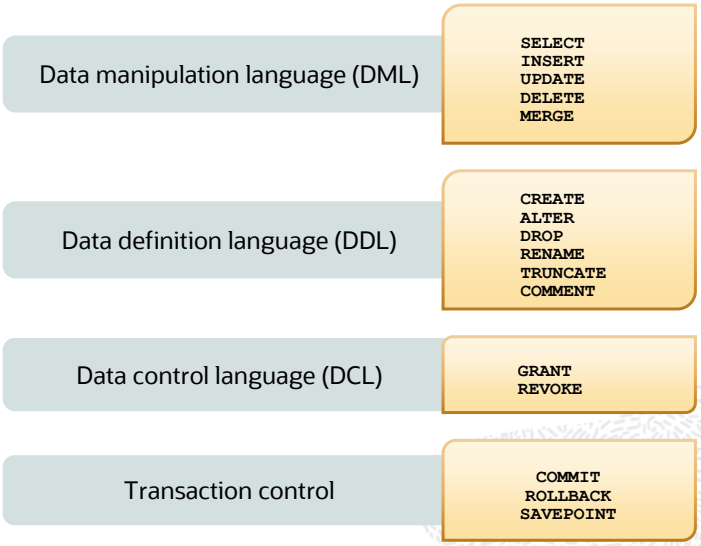
Using SQL benefits all types of users, including application programmers, database administrators, managers, and end users. The purpose of SQL is to provide an interface to a relational database such as Oracle Database. All SQL statements are instructions to the database. Some of the features of SQL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

Oracle Internal & Oracle Academy Use Only

SQL Statements Used in the Course



43

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.

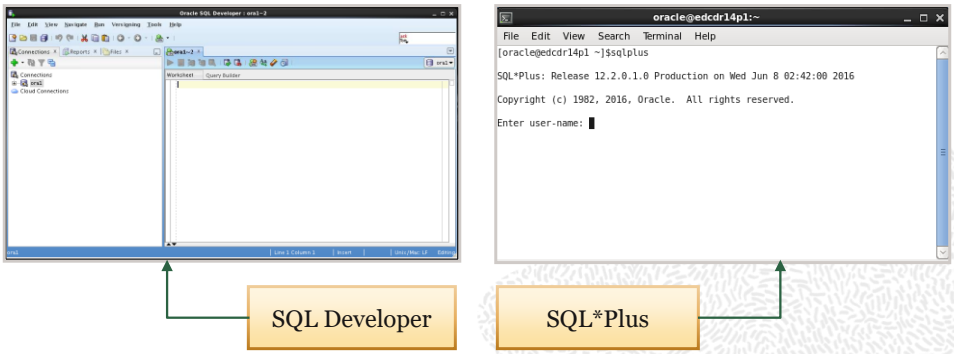
Statements	Description
SELECT INSERT UPDATE DELETE MERGE	Retrieve data from the database, enter new rows, change existing rows, and remove unwanted rows from tables in the database, respectively. Collectively known as *Data Manipulation Language* (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Set up, change, and remove data structures from tables. Collectively known as *Data Definition Language* (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.

Oracle Internal & Oracle Academy Use Only

Development Environments for SQL in Oracle

There are two Oracle development environments for this course:

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



44

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

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.

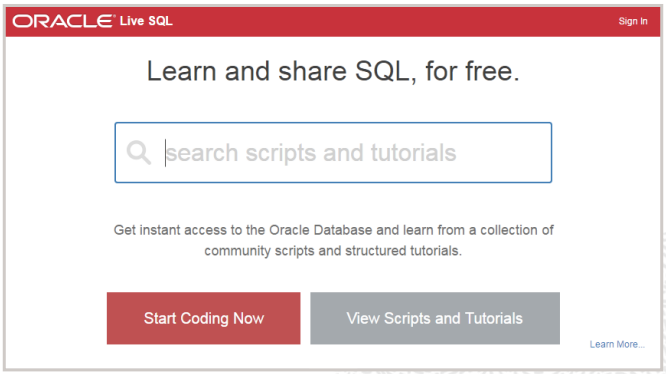
Notes

- 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.

Oracle Internal & Oracle Academy Use Only

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.



45

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.

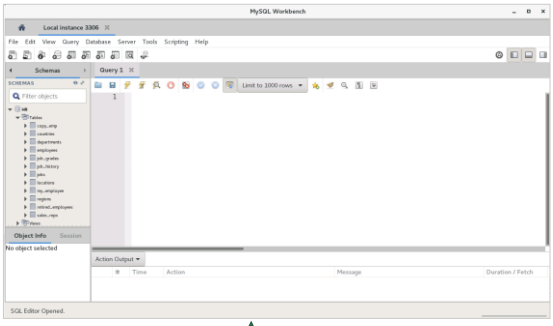
Note: Oracle Live SQL cannot be used to execute the lab exercises without the initial schema setup.

Oracle Internal & Oracle Academy Use Only

Development Environments for SQL in MySQL

There are two MySQL development environments available for this course:

- For this course, the primary tool is MySQL Workbench.
- mysql, the MySQL command-line tool, can be used instead.



46

There are various development environments for writing SQL statements. MySQL Workbench and the mysql command-line tool are commonly used for MySQL database development.

MySQL Workbench

This is a graphical interface for developing SQL. It provides features to view the database components and update values by navigating a graphical user interface in addition to entering SQL statements. In this course, MySQL Workbench is used to create and execute example SQL statements. You will use MySQL Workbench to perform the hands-on exercises.

`mysql` – the MySQL Command-Line Tool

If you like working with a command-line interface, you can use mysql, which is the command-line interpreter for the MySQL Database. It processes SQL statements and also MySQL-specific commands. It can also be used to run all SQL commands covered in this course.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 19c 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
- Oracle Database 19c SQL Documentation and Additional Resources

Oracle Internal & Oracle Academy Use Only

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

48

Navigate to https://docs.oracle.com/en/database/database.html to access the Oracle Database 19c documentation library.

Oracle Internal & Oracle Academy Use Only

Additional Resources for Oracle

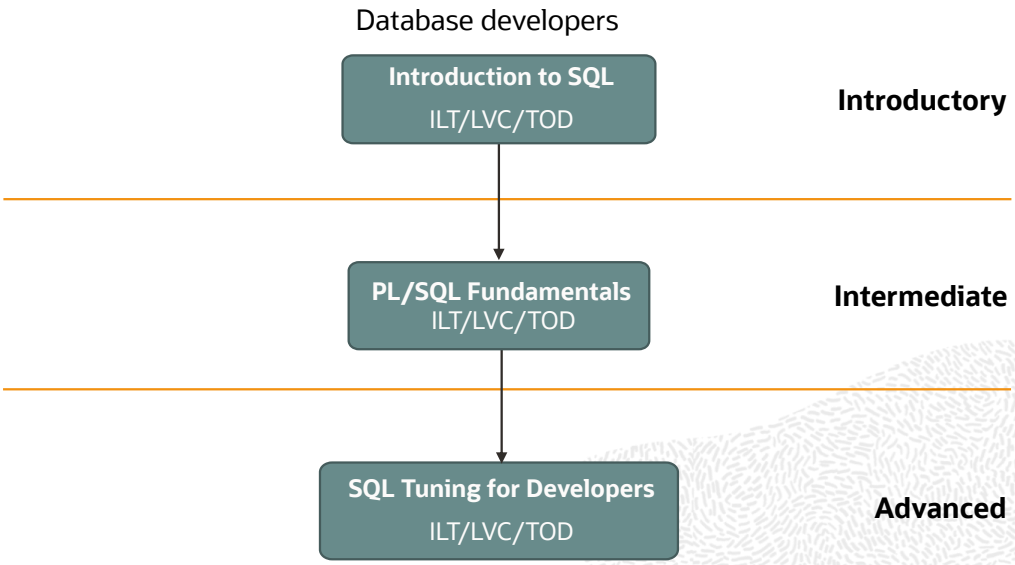
For additional information about Oracle Database 19c, refer to the following:

- Oracle Learning Library:
  - http://www.oracle.com/goto/oll
- Oracle Cloud:
  - cloud.oracle.com
- 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

49

Oracle Internal & Oracle Academy Use Only

Oracle University: Oracle SQL Training



50

Training Formats

- **Instructor-Led Training (ILT):** Delivered in a classroom with the instructor and students present at the same location and time
- **Live Virtual Class (LVC):** Delivered by using video and audio through a web-based delivery system (WebEx) in which geographically distributed instructor and students participate, interact, and collaborate in a virtual class environment
- **Training On Demand (TOD):** On-demand training that takes traditional classroom training (complete with all classroom content, including lectures, white boarding, and practice videos) and makes it available in a video-based, online format so that you can start your customized training at your convenience

For full details about Oracle SQL training options, go to http://education.oracle.com.

Oracle Internal & Oracle Academy Use Only

Oracle SQL Certification

The Oracle Certification Program validates your expertise in the following areas:

- Oracle Database SQL Certified Associate
- Oracle Database PL/SQL Certified Associate
- Oracle Database PL/SQL Certified Professional

Take the examination at a local Pearson VUE test center:

- At a time suitable to you
- In over 175 countries

51

For full details of SQL and other Oracle Certification options, visit:
http://education.oracle.com/certification.

Oracle Internal & Oracle Academy Use Only

MySQL Websites

- http://www.mysql.com:
  - Product information
  - Services (Training, Certification, Consulting, and Support)
  - White papers, webinars, and other resources
  - MySQL Enterprise Edition downloads (trial versions)
- http://dev.mysql.com:
  - Developer Zone (forums, articles, Planet MySQL, and more)
  - Documentation
  - Downloads
- https://github.com/mysql:
  - Source code for MySQL Server and other MySQL products

52

Download MySQL Community Edition general availability (GA) and development releases from the http://dev.mysql.com website. This site also hosts the online documentation, which is an extremely valuable resource for both beginners and expert users of MySQL, and includes several example databases. Download trial versions of the MySQL Enterprise Edition software, view detailed product information, and find out more about Oracle MySQL services at http://www.mysql.com.

Oracle Internal & Oracle Academy Use Only

MySQL Community Resources

- Mailing lists
- Forums
- Developer articles
- MySQL Newsletter (published monthly)
- Planet MySQL blogs
- Social media channels
  - Facebook, Twitter, and Google+
- Physical and virtual events, including:
  - Developer days
  - MySQL Tech Tours
  - Webinars
- Bug tracking
- GitHub repositories

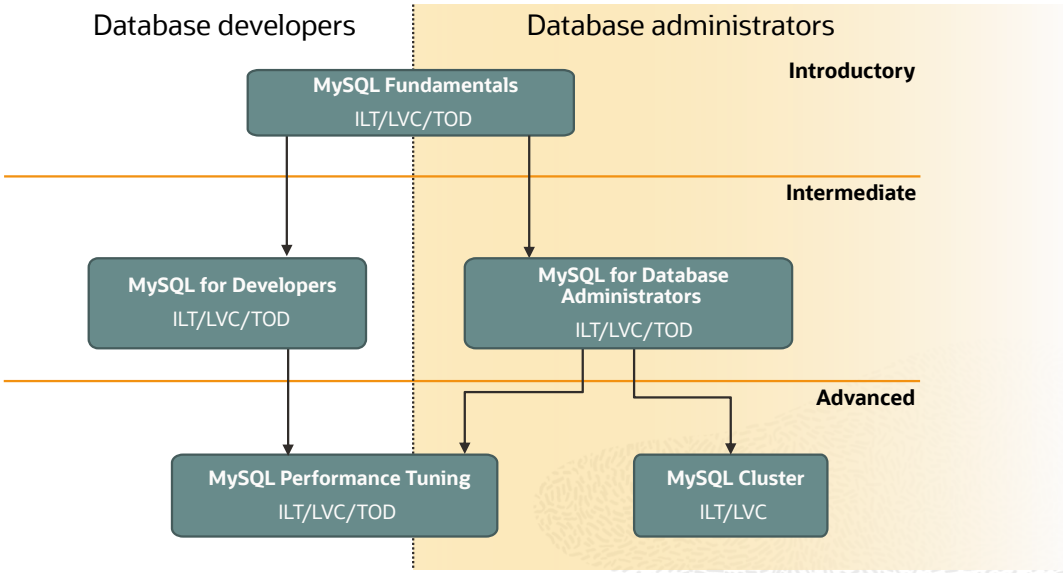
53

MySQL has a very large and active community. Engage with your peers, report bugs, share information, and discover what is happening in the world of MySQL by using the following resources:

- Mailing lists (http://lists.mysql.com)
- Forums (http://forums.mysql.com)
- Developer articles (http://forums.mysql.com
- MySQL Newsletter (http://www.mysql.com/news-and-events/newsletter/)
- Planet MySQL blogs (http://planet.mysql.com)
- Social media channels:
  - Facebook: http://facebook.com/mysql
  - Twitter: https://twitter.com/mysql_community and http://twitter.com/MySQL
- Physical and virtual events (http://www.mysql.com/news-and-events)
- Bug tracking (http://bugs.mysql.com)
- GitHub repositories

Oracle Internal & Oracle Academy Use Only

Oracle University: MySQL Training



54

Training Formats

- **Instructor-Led Training (ILT):** Delivered in a classroom with the instructor and students present at the same location and time
- **Live Virtual Class (LVC):** Delivered by using video and audio through a web-based delivery system (WebEx) in which geographically distributed instructor and students participate, interact, and collaborate in a virtual class environment
- **Training On Demand (TOD):** On-demand training that takes traditional classroom training (complete with all classroom content, including lectures, white boarding, and practice videos) and makes it available in a video-based, online format so that you can start your customized training at your convenience

For full details about MySQL training options, go to http://education.oracle.com/mysql.

Oracle Internal & Oracle Academy Use Only

MySQL Certification

The Oracle Certification Program validates your expertise in the following areas:

- Oracle Certified Professional: MySQL 5.6 Database Administrator
- Oracle Certified Professional: MySQL 5.6 Developer

Take the examination at a local Pearson VUE test center:

- At a time suitable to you
- In over 175 countries

55

For full details of MySQL and other Oracle Certification options, visit:
http://education.oracle.com/certification.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned about:

- The goals of the course
- The features of Oracle Database 19c
- 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

56

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.

Oracle Internal & Oracle Academy Use Only

Practice 1: Overview

This practice covers the following topics:

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

57

In this practice, you perform the following:

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

Note the following location for the lab files:

```
/home/oracle/labs/sql1_oracle/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.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

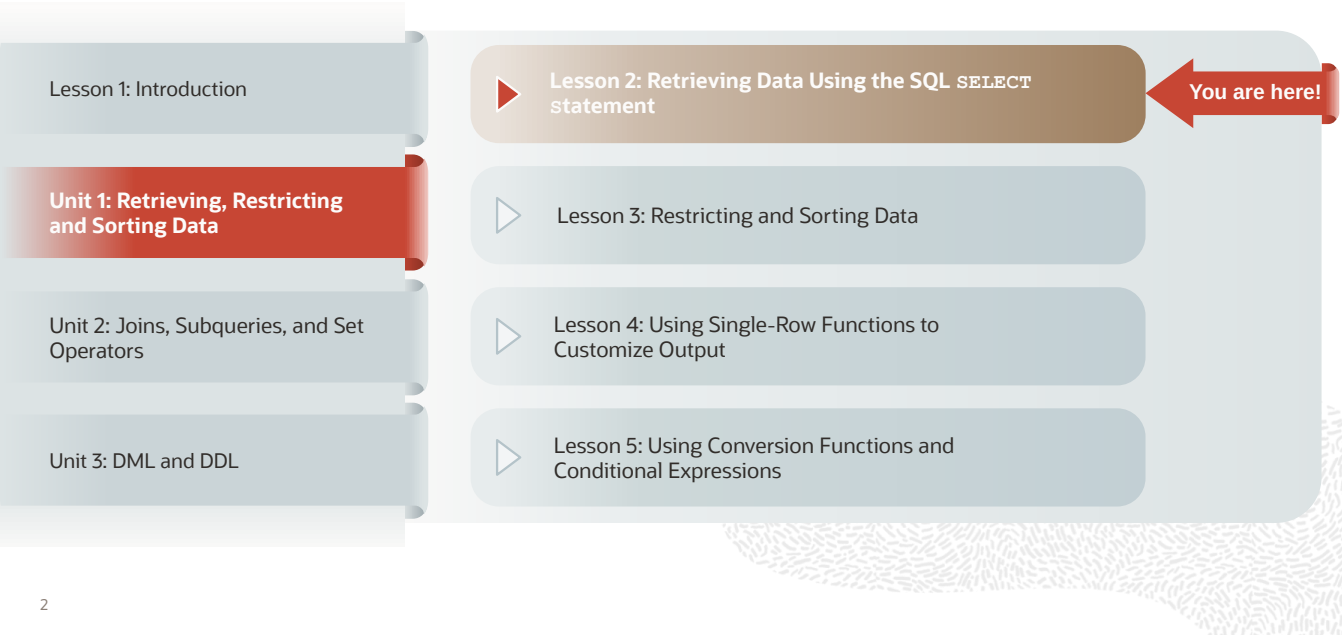
ORACLE

2

Retrieving Data Using the SQL SELECT Statement

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 1, you will learn how to query the data from tables, how to query selected records from tables, and also how to sort the data retrieved from the tables.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- List the capabilities of SQL `SELECT` statements
- Execute a basic `SELECT` statement

3

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.

Oracle Internal & Oracle Academy Use Only

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

4

Oracle Internal & Oracle Academy Use Only

HR Application Scenario

5

Alex, an HR manager in India, wants a report of all the employees in the Accounting department of the organization. This slide displays a mockup of the screens that the application might display in order to access the database. 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. You will not learn about the application used to develop the interface screens like those shown in the mockup in this slide. Those are covered in Oracle courses on application development like APEX and in the MySQL for Developers course.

Oracle Internal & Oracle Academy Use Only

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.

6

Writing SQL Statements

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

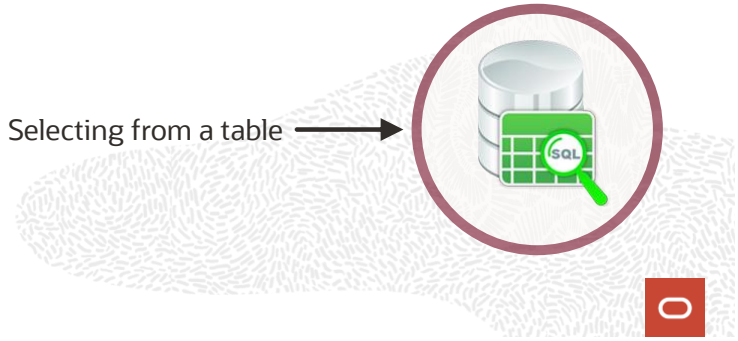
- SQL statements are not case-sensitive (unless indicated).
- SQL statements can be entered on one or many lines.
- Keywords cannot be split across lines or abbreviated.
- Clauses are usually placed on separate lines for readability and ease of editing.
- Indents should be used to make code more readable.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

Oracle Internal & Oracle Academy Use Only

Basic SELECT Statement

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

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



7

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
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

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.

Oracle Internal & Oracle Academy Use Only

Selecting All Columns

Oracle SQL Developer:

MySQL Workbench:

```
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

#	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
*	NULL	NULL	NULL	NULL

8

You can display all columns of data in a table by following the `SELECT` keyword with an asterisk (`*`). In the example in the slide, the `departments` table contains four columns: `department_id`, `department_name`, `manager_id`, and `location_id`. The table contains eight rows, one for each department.

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

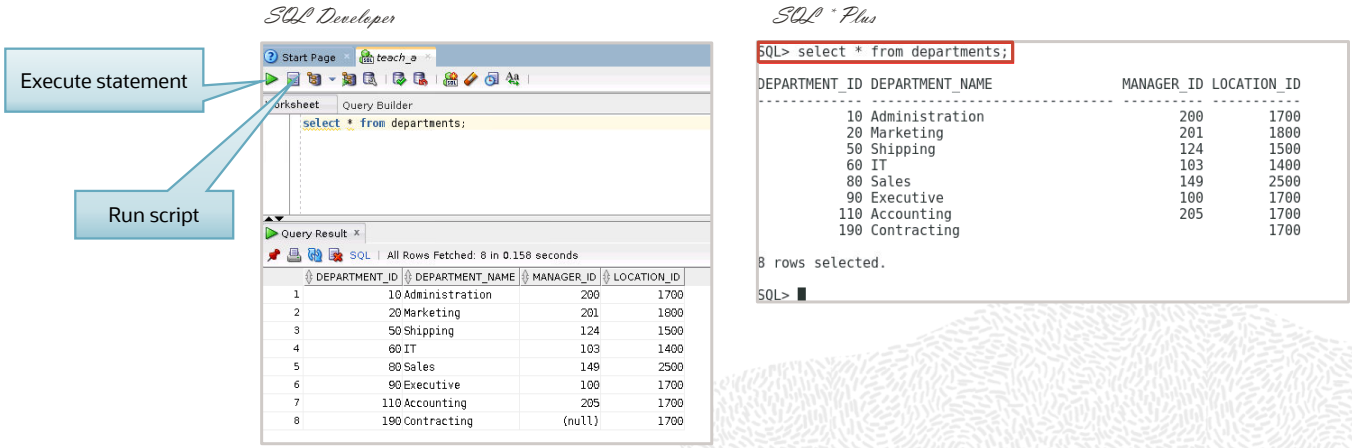
```
SELECT  department_id, department_name, manager_id, location_id
FROM    departments;
```

The examples in the slide show the output from executing the displayed statement in both Oracle SQL Developer and MySQL Workbench. Before considering other `SELECT` statements, the next few topics explain how to execute statements and what the output looks like in SQL Developer and MySQL Workbench.

Note: In this course, for examples on slides when both Oracle and MySQL use the same syntax, the outputs of both Oracle SQL Developer and MySQL Workbench appear. Your output should appear like the output of the database you are using. For topics where the databases use a different syntax or have other significant differences, separate slides are provided. You can skip over slides that do not apply to the database you are using.

Oracle Internal & Oracle Academy Use Only

Executing SQL Statements with Oracle SQL Developer and SQL*Plus



9

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:

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

Oracle Internal & Oracle Academy Use Only

Column Heading Defaults in SQL Developer and SQL*Plus

- 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

10

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:

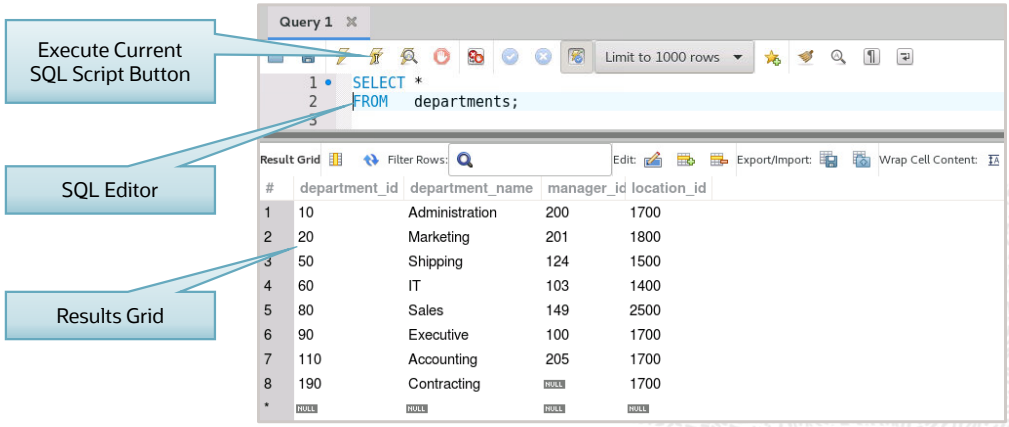
```
SELECT last_name, hire_date, salary
FROM   employees;
```

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

Oracle Internal & Oracle Academy Use Only

Executing SQL Statements in MySQL Workbench

Enter statements in the SQL Editor. To execute a single statement, place the cursor anywhere in the statement and click the **Execute Current SQL Script** button or press **Ctrl+Enter**. The results display in the Results Grid.



11

You can enter multiple statements in the SQL Editor. You can select one statement or multiple statements in the SQL Editor and click the **Execute SQL Script** button (to the left of the Execute Current SQL Script button) or press **Ctrl+Shift+Enter** to execute the selected statement or statements or execute all statements in the SQL Editor if nothing is selected.

By default, the headings of the results in the Result Grid are the column names, displayed left-aligned above the results. 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.

Oracle Internal & Oracle Academy Use Only

Executing SQL Statements in `mysql` Command-line Client

Enter statements in the `mysql` command-line client. Press **Enter** to continue a statement to another line. Terminate a statement with semicolon (`;`) and press **Enter** to execute the statement. Results display in a text table.

```
mysql> SELECT *
    -> FROM departments;
+---------------+----------------+------------+-------------+
| department_id | department_name | manager_id | location_id |
+---------------+----------------+------------+-------------+
|            10 | Administration  |        200 |        1700 |
|            20 | Marketing       |        201 |        1800 |
|            50 | Shipping        |        124 |        1500 |
|            60 | IT              |        103 |        1400 |
|            80 | Sales           |        149 |        2500 |
|            90 | Executive       |        100 |        1700 |
|           110 | Accounting      |        205 |        1700 |
|           190 | Contracting     |       NULL |        1700 |
+---------------+----------------+------------+-------------+
8 rows in set (0.00 sec)
```

12

The default prompt for `mysql` command line client is `mysql>`. When you press Enter before terminating the statement with a semicolon, the prompt becomes `->`. The MySQL examples in this course use MySQL Workbench, but you can use the `mysql` command-line client if you prefer a command-line tool.

By default, the headings of the results are the column names, displayed 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.

Oracle Internal & Oracle Academy Use Only

Selecting Specific Columns

Oracle SQL Developer:

MySQL Workbench:

```
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

#	department_id	location_id
1	60	1400
2	50	1500
3	10	1700
4	90	1700
5	110	1700
6	190	1700
7	20	1800
8	80	2500
*	NULL	NULL

13

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. The results from Oracle SQL Developer and MySQL Workbench are the same. The order of the results might differ unless you specify an order. You learn to specify an order for the results in the lesson titled “Restricting and Sorting Data.”

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 the location before department number (from left to right), you use the following statement:

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

Oracle Internal & Oracle Academy Use Only

Selecting from `dual` with Oracle Database

- `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	25-JUN-18

14

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;
```

Oracle Internal & Oracle Academy Use Only

Selecting Constant Expressions in MySQL

MySQL accepts the `FROM DUAL` clause but ignores it. The following statements are equivalent:

```
SELECT SYSDATE();
```

#	SYSDATE()
1	2018-08-17 18:24:44

```
SELECT SYSDATE()
FROM   DUAL;
```

#	SYSDATE()
1	2018-08-17 18:24:44

15

Selecting from the `dual` table is useful for computing a constant expression with the `SELECT` statement. It returns the value 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` either of the following equivalent statements:

```
SELECT  12*4/5+5*8
FROM dual;

SELECT  12*4/5+5*8;
```

MySQL does not create a dual table. In MySQL, the statement `SELECT * from dual;` will generate an error message because the `dual` table does not exist.

Oracle Internal & Oracle Academy Use Only

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

16

Oracle Internal & Oracle Academy Use Only

Arithmetic Expressions

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

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

17

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

Arithmetic Operators

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

Remember that you can use only the addition and subtraction operators with the `DATE` and `TIMESTAMP` data types.

Oracle Internal & Oracle Academy Use Only

Using Arithmetic Operators

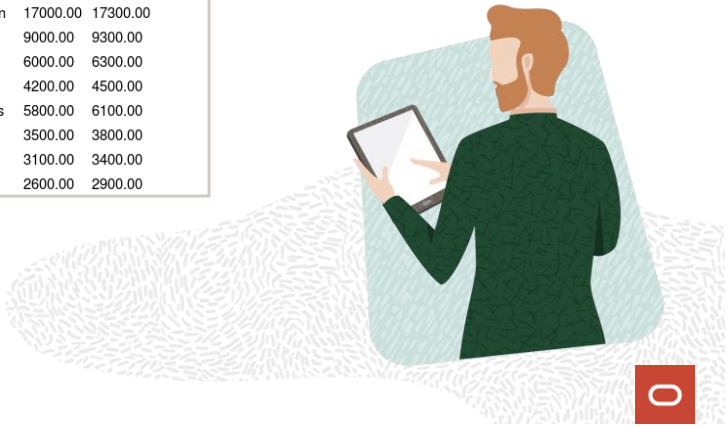
```
SELECT last_name, salary, salary + 300
FROM   employees;
```

	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

...

#	last_name	salary	salary + 300
1	King	24000.00	24300.00
2	Kochhar	17000.00	17300.00
3	De Haan	17000.00	17300.00
4	Hunold	9000.00	9300.00
5	Ernst	6000.00	6300.00
6	Lorentz	4200.00	4500.00
7	Mourgos	5800.00	6100.00
8	Rajs	3500.00	3800.00
9	Davies	3100.00	3400.00
10	Matos	2600.00	2900.00

...



18

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.

Oracle Internal & Oracle Academy Use Only

Operator Precedence

1

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

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

...

#	last_name	salary	12*salary+100
1	King	24000.00	288100.00
2	Kochhar	17000.00	204100.00
3	De Haan	17000.00	204100.00
4	Hunold	9000.00	108100.00

...

2

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

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

...

#	last_name	salary	12*(salary+100)
1	King	24000.00	289200.00
2	Kochhar	17000.00	205200.00
3	De Haan	17000.00	205200.00
4	Hunold	9000.00	109200.00

...

19

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

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

Using Parentheses

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

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

Oracle Internal & Oracle Academy Use Only

Defining a Null Value

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

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

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
3	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)

#	last_name	job_id	salary	commission_pc
1	King	AD_PRES	24000.00	NULL
2	Kochhar	AD_VP	17000.00	NULL
3	De Haan	AD_VP	17000.00	NULL
...				
12	Zlotkey	SA_MAN	10500.00	0.20
13	Abel	SA_REP	11000.00	0.30
14	Taylor	SA_REP	8600.00	0.20
15	Grant	SA_REP	7000.00	0.15
...				
18	Fay	MK_REP	6000.00	NULL
19	Higgins	AC_MGR	12008.00	NULL
20	Gietz	AC_ACC...	8300.00	NULL

20

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 example in the slide, 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, and MySQL Workbench displays `NULL` as a graphic in white on gray. The mysql command-line client displays the word `NULL`.

Oracle Internal & Oracle Academy Use Only

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

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

	LAST_NAME	12*SALARY*COMMISSION_PCT
1	King	(null)
2	Kochhar	(null)
3	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)

#	last_name	12*salary*commission_pct
1	King	NULL
2	Kochhar	NULL
3	De Haan	NULL
...		
12	Zlotkey	25200.0000
13	Abel	39600.0000
14	Taylor	20640.0000
15	Grant	12600.0000
...		
17	Hartstein	NULL
18	Fay	NULL
19	Higgins	NULL
20	Gietz	NULL

21

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

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

Oracle Internal & Oracle Academy Use Only

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

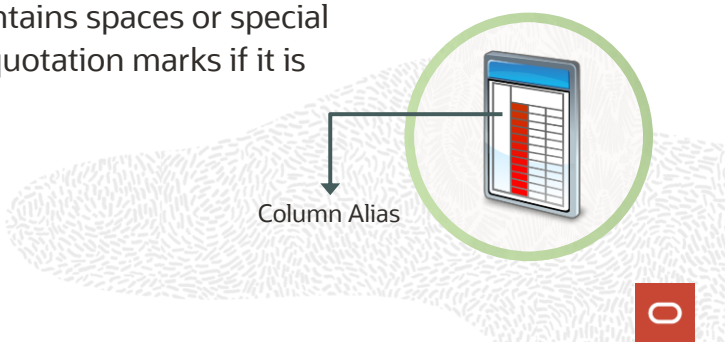
22

Oracle Internal & Oracle Academy Use Only

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. In Oracle, it requires double quotation marks if it is case-sensitive.



23

When displaying the result of a query, the results normally use the name of the selected column as the column heading. This heading might not be descriptive and, therefore, might 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 or the optional keyword `AS` as a separator between the column name and the alias.

In Oracle database, by default, alias headings appear in uppercase, so enclose a case-sensitive alias in double quotation marks (`" "`). In Oracle database, if the alias contains spaces or special characters (such as –, `!`, _), enclose the alias in double quotation marks (`" "`).

In MySQL, if the alias contains spaces or special characters, quote the alias by enclosing it in backticks (` ` `), single quotation marks (`' '`), or double quotation marks (`" "`). In MySQL, alias headings are case-sensitive. That is, they appear in the case specified, even if they are not quoted with backticks (` ` `), single quotation marks (`' '`), or double quotation marks (`" "`).

Oracle Internal & Oracle Academy Use Only

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)

...

#	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

...

#	Name	Annual Salary
1	King	288000.00
2	Kochhar	204000.00
3	De Haan	204000.00
4	Hunold	108000.00

...

24

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

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

Note: In MySQL the second example does not require the double quotation marks for ‘Name’. It would be equivalent as the following:

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

Oracle Internal & Oracle Academy Use Only

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

25

Oracle Internal & Oracle Academy Use Only

Concatenation Operator in Oracle

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

...

26

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

In Oracle databases, if you concatenate a null value with a character string, the result is a character string. `LAST_NAME || NULL` results in `LAST_NAME`.

Oracle Internal & Oracle Academy Use Only

Concatenation Function in MySQL – CONCAT()

The CONCAT() function:

- Links columns or character strings to other columns
- Is a function that concatenates the values provided to it
- Creates a resultant column that is a character expression

```
SELECT    CONCAT(last_name, job_id) AS "Employees"
FROM employees;
```

#	Employees
1	KingAD_PRES
2	KochharAD_VP
3	De HaanAD_VP
4	HunoldIT_PROG
5	ErnstIT_PROG
6	LorentzIT_PROG
7	MourgosST_MAN
8	RajsST_CLERK

...

27

You can link columns to other columns, arithmetic expressions, or constant values to create a character expression by using CONCAT(). Columns names, separated by commas, 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, with no space between them.

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

Null Values with the Concatenation Operator

In MySQL, if you use the CONCAT() function to concatenate a null value with a character string, the result is NULL.

Oracle Internal & Oracle Academy Use Only

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.

28

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

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

MySQL accepts either single quotation marks (`' '`) or double quotation marks (`" "`), unless `ANSI_QUOTES` is enabled for the `sql_mode` system variable. Setting system variables is beyond the scope of this course, so examples of literal character strings in this course use single quotation marks.

Oracle Internal & Oracle Academy Use Only

Using Literal Character Strings in Oracle

```
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

29

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;
```

Oracle Internal & Oracle Academy Use Only

Using Literal Character Strings in MySQL

```
SELECT CONCAT(last_name,' is a ', job_id)
       AS 'Employee Details'
FROM   employees;
```

#	Employee Details
1	King is a AD_PRES
2	Kochhar is a AD_VP
3	De Haan is a AD_VP
4	Hunold is a IT_PROG
5	Ernst is a IT_PROG
6	Lorentz is a IT_PROG
7	Mourgos is a ST_MAN
8	Rajs is a ST_CLERK
9	Davies is a ST_CLERK
10	Matos is a ST_CLERK

...

30

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 CONCAT(last_name, ': 1 Month salary = ', salary) AS Monthly
FROM   employees;
```

Oracle Internal & Oracle Academy Use Only

Alternative Quote (q) Operator in Oracle

- 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:

31

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

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

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

Oracle Internal & Oracle Academy Use Only

Including a Single Quotation Mark in a String with an Escape Sequence in MySQL

- To indicate a quotation mark is to be included in a string, use the `\'` escape sequence.

```
SELECT CONCAT(department_name,
  ' 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	NULL

32

Many SQL statements use character literals in expressions or conditions. If the literal itself contains a single quotation mark, you can use the (\) escape sequence to indicate that the quotation mark is part of the string.

In the example shown, the string contains a single quotation mark, which is normally interpreted as a delimiter of a character string. By using the `\'` escape sequence, however, the single quotation mark is included as part of the string. MySQL supports a number of other escape sequences.

Oracle Internal & Oracle Academy Use Only

Duplicate Rows

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

```
SELECT department_id
FROM   employees;
```

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

...

#	department_id
1	NULL
2	10
3	20
4	20
5	50
6	50
7	50
8	50
9	50

...

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

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

#	department_id
1	NULL
2	10
3	20
4	50
5	60
6	80
7	90
8	110

33

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

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

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

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

Oracle Internal & Oracle Academy Use Only

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

Oracle Internal & Oracle Academy Use Only

Displaying Table Structure by Using the DESCRIBE Command

Syntax:

```
DESCRIBE tablename
```

Example:

```
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)
```

#	Field	Type	Null	Key	Default	Extra
1	employee_id	int(11)	NO	PRI	NULL	
2	first_name	varchar(20)	YES		NULL	
3	last_name	varchar(25)	NO	MUL	NULL	
4	email	varchar(25)	NO	UNI	NULL	
5	phone_number	varchar(20)	YES		NULL	
6	hire_date	date	NO		NULL	
7	job_id	varchar(10)	NO	MUL	NULL	
8	salary	decimal(8,2)	YES		NULL	
9	commission_pct	decimal(2,2)	YES		NULL	
10	manager_id	int(11)	YES	MUL	NULL	
11	department_id	int(11)	YES	MUL	NULL	

35

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

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` in SQL Developer or `NO` in MySQL Workbench indicates that a column must contain data. *Type* displays the data type for a column.

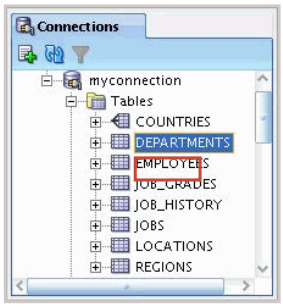
Data types are explained in the lesson titled “Introduction to Data Definition Language.”

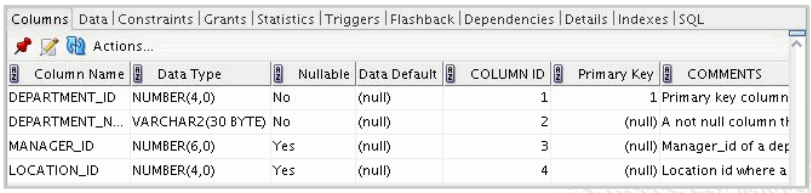
Data Type	Description
`NUMBER(p,s)`	Number value having a maximum number of digits *p*, with *s* digits to the right of the decimal point
`VARCHAR2(s)`	Variable-length character value of maximum size *s*
`DATE`	Date and time value between January 1, 4712 B.C. and December 31, A.D. 9999

Oracle Internal & Oracle Academy Use Only

Displaying Table Structure by Using Oracle SQL Developer

- 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.





36

You can display the structure of a table in Oracle SQL Developer 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.

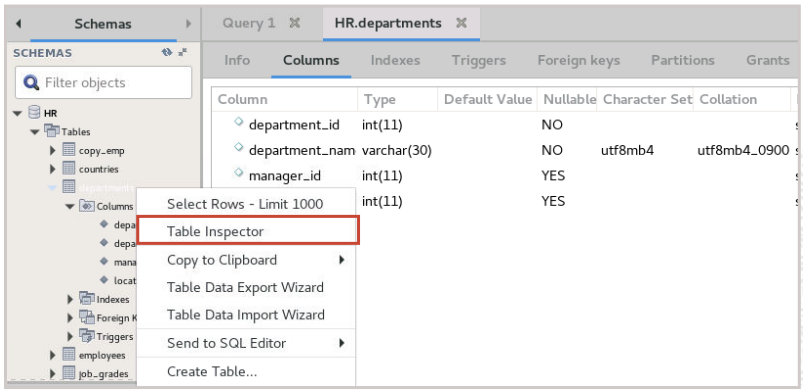
Using the SQL Developer GUI interface, you can select the table in the Connections tree and use the Columns tab to view the table structure.

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

Oracle Internal & Oracle Academy Use Only

Displaying Table Structure by Using MySQL Workbench

- Use the `DESCRIBE` command to display the structure of a table.
- Alternatively, right-click the table in the Navigator and select **Table Inspector** from the menu. Select the **Columns** tab.



37

You can display the structure of a table in MySQL Workbench 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 Nullable column indicates `NO`).

Using the SQL Developer GUI interface, you can right-click the table in the Navigator and select **Table Inspector** from the menu. Then select the **Columns** tab.

Note: `DESCRIBE` is abbreviated as `DESC`.

Oracle Internal & Oracle Academy Use Only

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

38

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

Oracle Internal & Oracle Academy Use Only

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

39

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

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

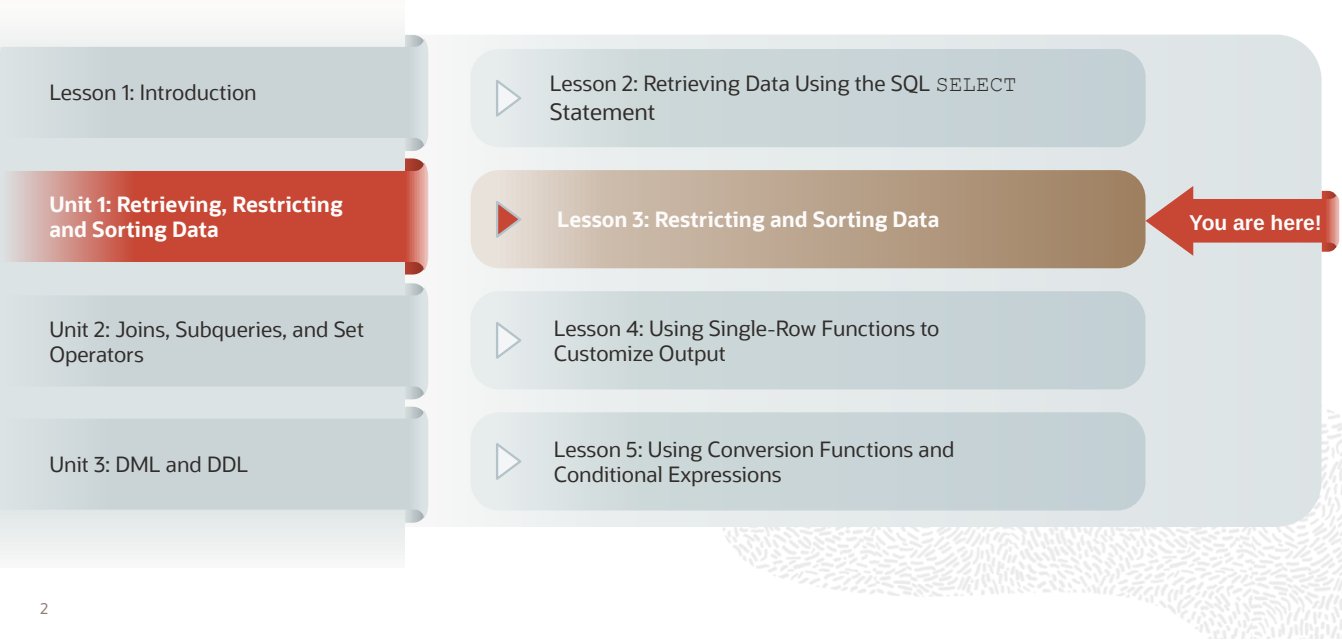
ORACLE

3

Restricting and Sorting Data

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 1, you will learn how to query the data from tables, how to query selected records from tables, and also how to sort the data retrieved from the tables.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query

3

When retrieving data from a database, you may need to:

- 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.

Oracle Internal & Oracle Academy Use Only

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 in Oracle
- Assigning values to variables

4

Oracle Internal & Oracle Academy Use Only

Limiting Rows by Using a Selection

```
SELECT employee_id, last_name, job_id, department_id
FROM 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

...

#	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

...

What if you want to retrieve all employees in department 90, but not other departments?

	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

#	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

5

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

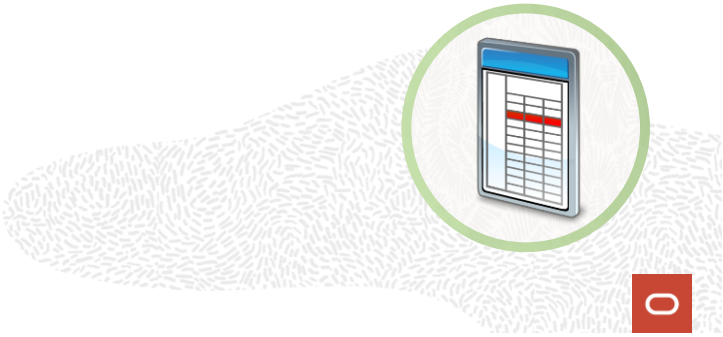
Oracle Internal & Oracle Academy Use Only

Limiting 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.



6

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	Restricts the query to rows that meet a condition
logical expression	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 KNOWN.

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

Oracle Internal & Oracle Academy Use Only

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

#	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
*	NULL	NULL	NULL	NULL

7

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

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

Oracle Internal & Oracle Academy Use Only

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` in Oracle databases and `YYYY-MM-DD` in MySQL.

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

	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	Whalen	AD_ASST	10

#	last_name	job_id	department_id
1	Whalen	AD_ASST	10

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

	LAST_NAME
1	Rajs

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = '2011-10-17' ;
```

#	last_name	hire_date
1	Rajs	2011-10-17

8

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 and MySQL 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 in Oracle and `YYYY-MM-DD` in MySQL.

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.

Oracle Internal & Oracle Academy Use Only

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

9

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.

Oracle Internal & Oracle Academy Use Only

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

#	last_name	salary
1	Matos	2600.00
2	Vargas	2500.00

```
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

#	employee_id	first_name	last_name	email	phone_numbe	hire_date	job_id	salary	commission_pc	manager_id	department_id
1	174	Ellen	Abel	EABEL	011.44.1644....	2012-05-11	SA_REP	11000.00	0.30	149	80
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

10

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

In the second example, the `SELECT` statement retrieves all rows where the last name is `Abel`. Because * is used in the `SELECT` statement, all fields from the employees table appear in the result set.

Oracle Internal & Oracle Academy Use Only

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

#	last_name	salary
1	Rajs	3500.00
2	Davies	3100.00
3	Matos	2600.00
4	Vargas	2500.00

11

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';
```

Oracle Internal & Oracle Academy Use Only

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

#	employee_id	last_name	salary	manager_id
1	101	Kochhar	17000.00	100
2	102	De Haan	17000.00	100
3	124	Mourgos	5800.00	100
4	149	Zlotkey	10500.00	100
5	201	Hartstein	13000.00	100
6	200	Whalen	4400.00	101
7	205	Higgins	12008.00	101
8	202	Fay	6000.00	201
*	NULL	NULL	NULL	NULL

12

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');
```

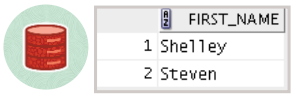
If characters or dates are used in a list, they must be enclosed within single quotation marks ('').

Oracle Internal & Oracle Academy Use Only

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
WHEREfirst_name LIKE 'S%' ;
```



	FIRST_NAME
1	Shelley
2	Steven

#	first_name
1	Shelley
2	Steven

13

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

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

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

The LIKE operator can be used as a shortcut for some BETWEEN comparisons. In Oracle, 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'
```

In MySQL, 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 '2015%';
```

Oracle Internal & Oracle Academy Use Only

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

#	last_name
1	Kochhar
2	Lorentz
3	Mourgos

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

14

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, in Oracle, 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 '\';
```

To do the same thing in MySQL, use the following statement:

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

Oracle Internal & Oracle Academy Use Only

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
1	King	(null)

#	last_name	manager_id
1	King	NULL

15

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;
```

Oracle Internal & Oracle Academy Use Only

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 both component conditions are true
OR	Returns TRUE if either component condition is true
NOT	Returns TRUE if the condition is false

16

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

Three logical operators are available in SQL:

- `AND`
- `OR`
- `NOT`

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

Oracle Internal & Oracle Academy Use Only

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

#	employee_id	last_name	job_id	salary
1	149	Zlotkey	SA_MAN	10500.00
2	201	Hartstein	MK_MAN	13000.00
*	NULL	NULL	NULL	NULL

17

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

Oracle Internal & Oracle Academy Use Only

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

#	employee_id	last_name	job_id	salary
1	100	King	AD_PRES	24000.00
2	101	Kochhar	AD_VP	17000.00
3	102	De Haan	AD_VP	17000.00
4	124	Mourgos	ST_MAN	5800.00
5	149	Zlotkey	SA_MAN	10500.00
6	174	Abel	SA_REP	11000.00
7	201	Hartstein	MK_MAN	13000.00
8	205	Higgins	AC_MGR	12008.00
*	NULL	NULL	NULL	NULL

18

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

Oracle Internal & Oracle Academy Use Only

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

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

19

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

Oracle Internal & Oracle Academy Use Only

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 in Oracle
- Assigning values to variables

20

Oracle Internal & Oracle Academy Use Only

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.

21

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

Oracle Internal & Oracle Academy Use Only

Rules of Precedence

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

#	last_name	department_id	salary
1	Hunold	60	9000.00
2	Ernst	60	6000.00
3	Lorentz	60	4200.00
4	Zlotkey	80	10500.00
5	Abel	80	11000.00

2

```
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

#	last_name	department_id	salary
1	Zlotkey	80	10500.00
2	Abel	80	11000.00

22

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."

Oracle Internal & Oracle Academy Use Only

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 in Oracle
- Assigning values to variables

23

Oracle Internal & Oracle Academy Use Only

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

...

#	last_name	job_id	department_id	hire_date
1	De Haan	AD_VP	90	2009-01-13
2	Kochhar	AD_VP	90	2009-09-21
3	Gietz	AC_ACCOUNT	110	2010-06-07
4	Higgins	AC_MGR	110	2010-06-07
5	King	AD_PRES	90	2011-06-17
6	Whalen	AD_ASST	10	2011-09-17
7	Rajs	ST_CLERK	50	2011-10-17

...

24

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. Rows are first sorted 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.

Oracle Internal & Oracle Academy Use Only

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

25

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 in Oracle and 1992-01-01 before 1995-01-01 in MySQL).
- Character values are displayed in the alphabetical order (for example, “A” first and “Z” last).
- In Oracle, null values are displayed last for ascending sequences and first for descending sequences. In MySQL, null values are displayed first for ascending sequences and last 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 example in the slide sorts the data by annual salary.

Note: In Oracle, 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. Those options are not valid in MySQL.

Oracle Internal & Oracle Academy Use Only

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

26

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

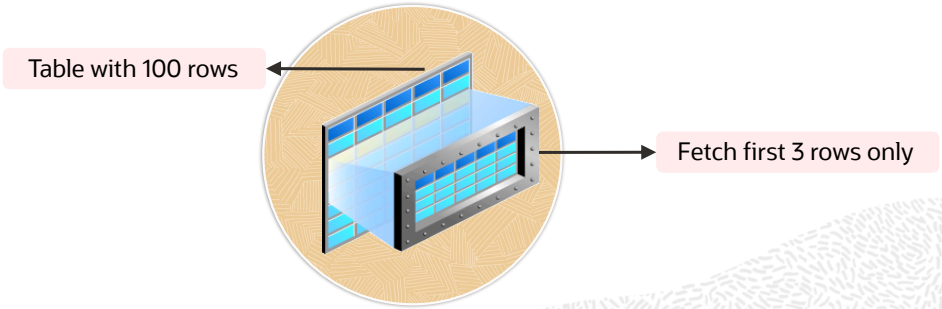
- Limiting rows with:
  - The `WHERE` clause
  - The comparison operators using `=`, `<=`, `BETWEEN`, `IN`, `LIKE`, and `NULL` conditions
  - Logical conditions using `AND`, `OR`, and `NOT` operators
- Rules of precedence for operators in an expression
- Sorting rows using the `ORDER BY` clause
- SQL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

27

Oracle Internal & Oracle Academy Use Only

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.



28

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

With the 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.

The SQL row limiting clause has certain limitations. These limitations are explained in the lesson titled "Managing Tables Using DML Statements and Transactions."

Oracle Internal & Oracle Academy Use Only

Using SQL Row Limiting Clause in a Query in Oracle

You specify `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 }]
```

29

You specify `row_limiting_clause` in the SQL `SELECT` statement by placing it after the `ORDER BY` clause. Note that an `ORDER BY` clause is required if you want to sort the rows for consistency.

- `OFFSET`: Use this clause to specify the number of rows to skip before row limiting begins. The value for offset must be a number. If you specify a negative number, offset is treated as 0. If you specify `NULL` or a number greater than or equal to the number of rows that are returned by the query, 0 rows are returned.
- `ROW` | `ROWS`: Use these keywords interchangeably. They are provided for semantic clarity.
- `FETCH`: Use this clause to specify the number of rows or percentage of rows to return.
- `FIRST` | `NEXT`: Use these keywords interchangeably. They are provided for semantic clarity.
- `row_count` | *percent* `PERCENT`: Use `row_count` to specify the number of rows to return. Use *percent* `PERCENT` to specify the percentage of the total number of selected rows to return. The value for percent must be a number.
- `ONLY` | `WITH TIES`: Specify `ONLY` to return exactly the specified number of rows or percentage of rows. Specify `WITH TIES` to return additional rows with the same sort key as the last row fetched. You must specify the `ORDER BY` clause for additional rows to be returned.

Oracle Internal & Oracle Academy Use Only

SQL Row Limiting Clause: Example in Oracle

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

Script Output | Query Result

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

30

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.

Oracle Internal & Oracle Academy Use Only

Using SQL Row Limiting Clause in a Query in MySQL

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 …  ]
 [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

31

You specify the row limiting clause in the SQL `SELECT` statement by placing it after the `ORDER BY` clause. Note that an `ORDER BY` clause is required if you want consistent output. Without an `ORDER BY` clause, the first rows are not predictable. MySQL uses either of the following syntaxes in the row limiting clause.

- Use the `LIMIT` keyword followed by a nonnegative numeric integer for the maximum number of rows, followed by the `OFFSET` keyword, followed by the offset of the first row to return. The offset of the initial row is 0. For example:
  - **`LIMIT 7 OFFSET 5`** returns 7 rows starting at row 6, that is rows 6 - 12.
- Use the `LIMIT` keyword, followed by either one or two nonnegative numeric arguments separated by a comma. With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0. For example:
  - **`LIMIT 5,7`** returns 7 rows starting at row 6, that is, rows 6 - 12. Note that this is the same as `LIMIT 7 OFFSET 5`.
  - **`LIMIT 0,7`** returns the first 7 rows, that is, rows 1 - 7
  - **`LIMIT 7`** returns the first 7 rows, that is, rows 1 - 7

Oracle Internal & Oracle Academy Use Only

SQL Row Limiting Clause: Example in MySQL

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
LIMIT 7;
```

#	employee_id	first_name
1	100	Steven
2	101	Neena
3	102	Lex
4	103	Alexander
5	104	Bruce
6	107	Diana
7	124	Kevin
*	NULL	NULL

```
SELECT employee_id, first_name
FROM employees
ORDER BY employee_id
LIMIT 7 OFFSET 5;
```

#	employee_id	first_name
1	107	Diana
2	124	Kevin
3	141	Trenna
4	142	Curtis
5	143	Randall
6	144	Peter
7	149	Eleni
*	NULL	NULL

32

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

The second example in the slide returns seven employees, starting with the sixth employee after sorting the rows in ascending order of the `employee_id`. For the second example, `LIMIT 5,7` would be equivalent to `LIMIT 7 OFFSET 5`.

Note: If `employee_id` is assigned sequentially by the date when the employee joined the organization, these examples give us the top 7 employees and then employees 6-12, all in terms of seniority.

Oracle Internal & Oracle Academy Use Only

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 in Oracle
- Assigning values to variables

33

Oracle Internal & Oracle Academy Use Only

Substitution Variables in Oracle

34

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 in Oracle.

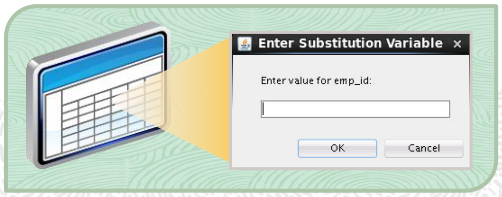
By using a **substitution variable** in place of the exact values in the `WHERE` clause, you can run the same query for different values.

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.

Oracle Internal & Oracle Academy Use Only

Substitution Variables in Oracle

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



35

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

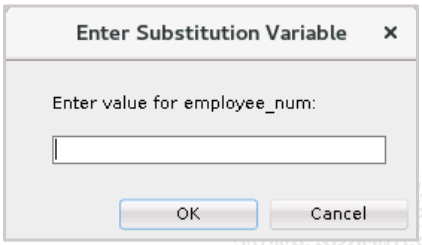
Note: Both Oracle SQL Developer and SQL*Plus support substitution variables and the `DEFINE/UNDEFINE` commands.

Oracle Internal & Oracle Academy Use Only

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 ;
```



36

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.

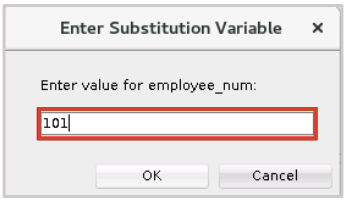
Notation	Description
&user_variable	Indicates a variable in a SQL statement; if the variable does not exist, SQL*Plus or SQL Developer prompts the user for a value (the new variable is discarded after it is used.)

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.

Oracle Internal & Oracle Academy Use Only

Using the Single-Ampersand Substitution Variable



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

37

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

After you enter a value and click the OK button, the results are displayed.

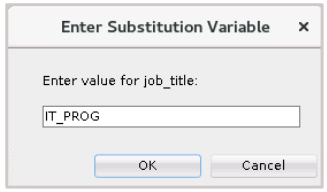
Note that if the substitution variable is referenced twice, even in the same command, then you are prompted twice. Different values can be entered at each prompt.

Oracle Internal & Oracle Academy Use Only

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

38

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

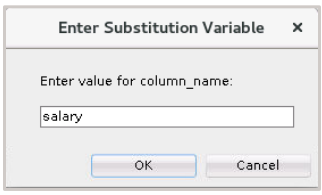
Enclose the variable with single quotation marks within the SQL statement itself.

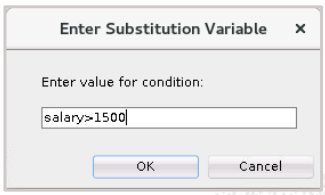
The slide shows a query to retrieve the employee names, department numbers, and annual salaries of all employees based on the job title value of the SQL Developer substitution variable.

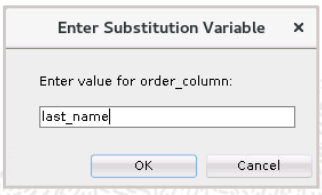
Oracle Internal & Oracle Academy Use Only

Specifying Column Names, Expressions, and Text

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







39

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.

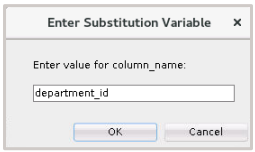
Note: A substitution variable can be used anywhere in the `SELECT` statement.

Oracle Internal & Oracle Academy Use Only

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 ;
```



	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

...

40

You can use the double-ampersand (&&) substitution variable if you want to reuse the variable value without prompting the user each time. The user sees the prompt for the value only once.

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

SQL Developer stores the value that is supplied by using the `DEFINE` command; it uses it again whenever you reference the variable name. After a user variable is in place, you need to use the `UNDEFINE` command to delete it:

```
UNDEFINE column_name;
```

Oracle Internal & Oracle Academy Use Only

Using the Ampersand Substitution Variable in SQL*Plus

```
SQL> SELECT employee_id, last_name, salary, department_id
  2  from employees
  3  where employee_id = &employee_num;
Enter value for employee_num: 101
```

```
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>
```

41

The example in the slide creates a SQL*Plus substitution variable for an employee number. When the statement is executed, SQL*Plus prompts you for an employee number and then displays the employee number, last name, salary, and department number for that employee.

Oracle Internal & Oracle Academy Use Only

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 in Oracle
- Assigning values to variables

42

Oracle Internal & Oracle Academy Use Only

Using the DEFINE Command in Oracle

- 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

43

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

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

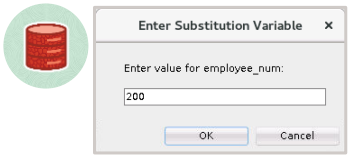
The EMPLOYEE_NUM substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

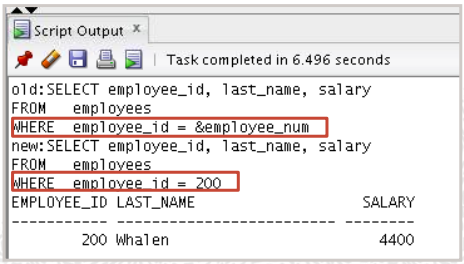
Oracle Internal & Oracle Academy Use Only

Using the VERIFY Command in Oracle

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

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





44

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.

Oracle Internal & Oracle Academy Use Only

Using the SET Statement in MySQL

- Use the SET statement to create a user-defined variable and assign a value to it.
- User-defined variables are entered as *@var_name*.

```
SET @employee_num = 200;

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

#	employee_id	last_name	salary	department_id
1	200	Whalen	4400.00	10
*	NULL	NULL	NULL	NULL

45

The example shown in the slide creates a user-defined variable for an employee number by using the SET command. At run time, the variable value is automatically substituted in the SELECT statement, which displays the employee number, name, salary, and department number for that employee.

The employee_num variable is specific to the session and remains assigned until the user assigns a different value or exits the session.

Oracle Internal & Oracle Academy Use Only

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

46

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

Oracle Internal & Oracle Academy Use Only

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

47

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.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

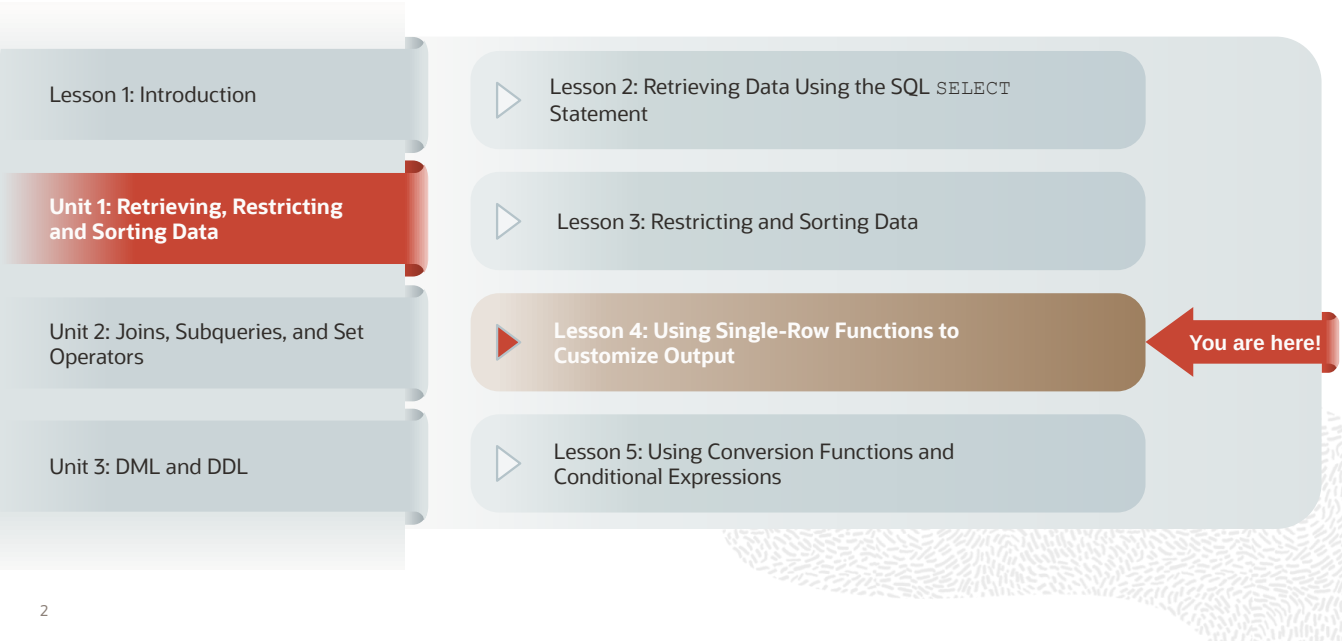
ORACLE

4

Using Single-Row Functions to Customize Output

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 1, you will learn how to query the data from tables, how to query selected records from tables, and also how to sort the data retrieved from the tables.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

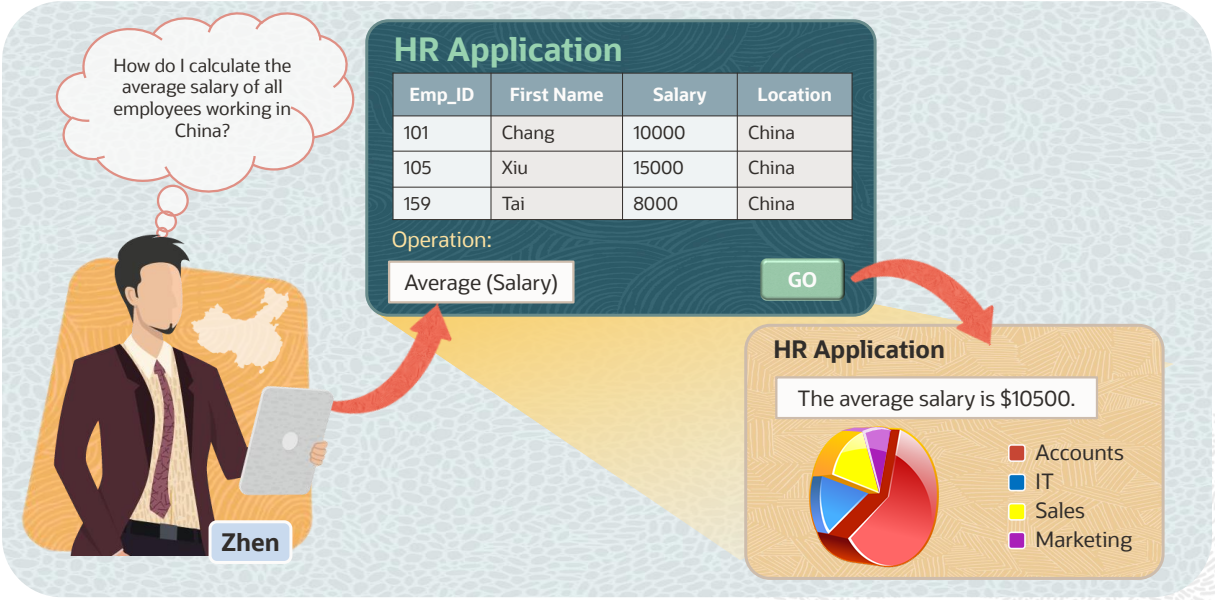
- Describe the various types of functions available in SQL
- Use the character, number, and date functions in SELECT statements

3

Functions make the basic query block more powerful, and they are used to manipulate data values. This is the first of two lessons that explore functions. It focuses on single-row character, number, and date functions.

Oracle Internal & Oracle Academy Use Only

HR Application Scenario



4

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.

Oracle Internal & Oracle Academy Use Only

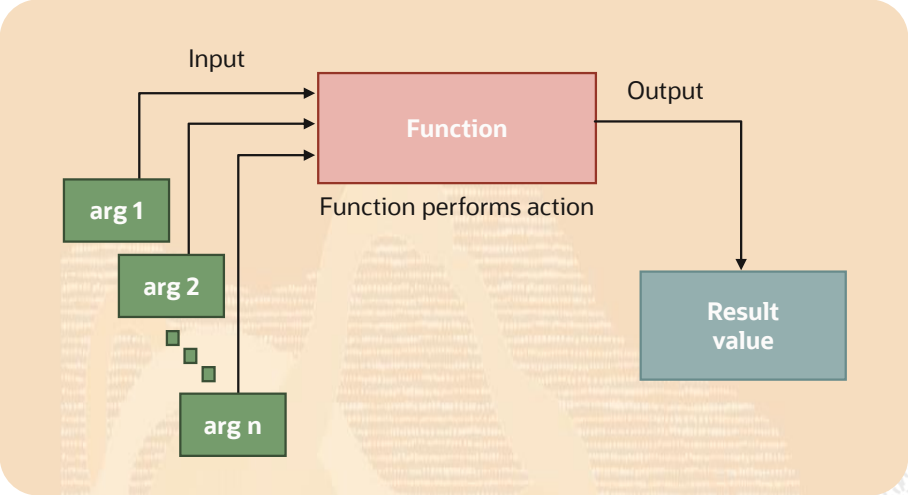
Lesson Agenda

- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

5

Oracle Internal & Oracle Academy Use Only

SQL Functions



6

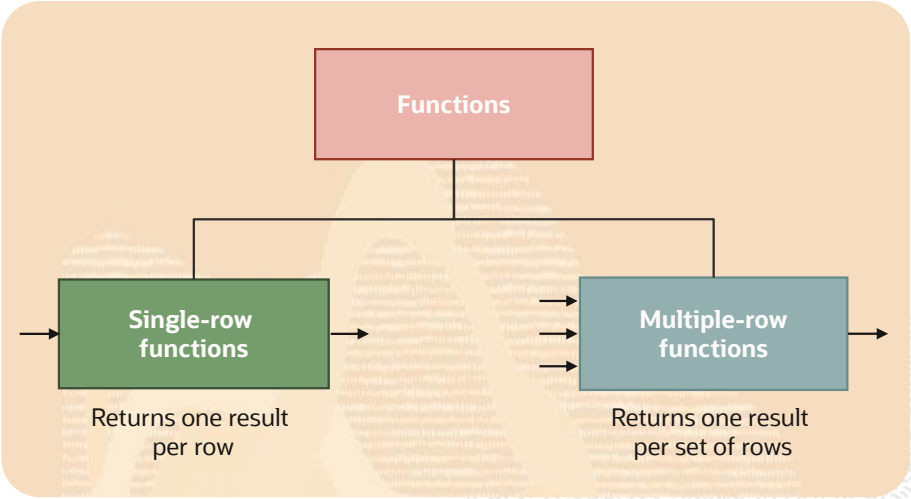
Functions are a very powerful feature of SQL. You can use functions to:

- 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.

Oracle Internal & Oracle Academy Use Only

Two Types of SQL Functions



7

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: There are many other functions available in both Oracle and MySQL databases. Not all can be covered in this course. This course explains how functions work and provides examples of some of the most commonly used function. Check the database documentation for other functions you might need for your application.

Oracle Internal & Oracle Academy Use Only

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
- Might modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

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

8

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

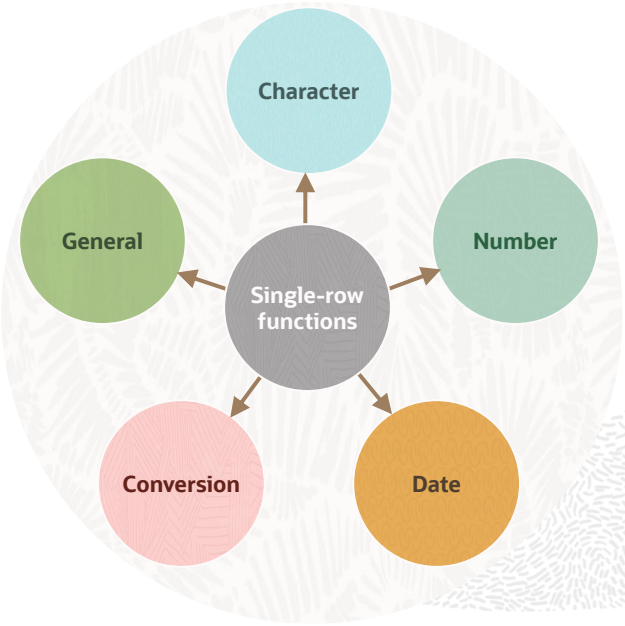
In the syntax:

function_name	Is the name of the function
arg1, arg2	Is any argument to be used by the function. This can be represented by a column name or expression.

Note: In MySQL, by default there must not be any space between the function name and the parenthesis for the arguments.

Oracle Internal & Oracle Academy Use Only

Single-Row Functions



9

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.
- **JSON functions:** These functions allow you to query and generate JSON data.

Oracle Internal & Oracle Academy Use Only

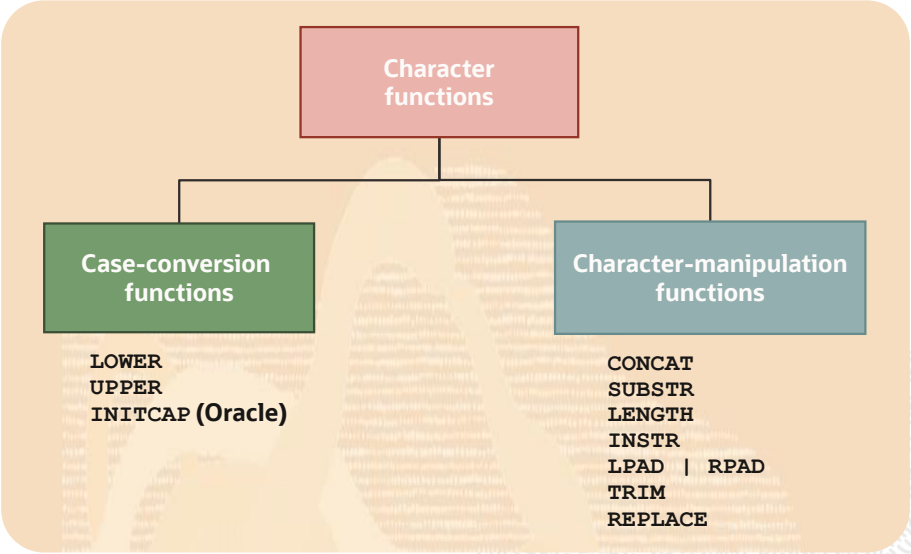
Lesson Agenda

- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

10

Oracle Internal & Oracle Academy Use Only

Character Functions



11

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(`*`column`*`|`*`expression`*`)`	Converts alpha character values to lowercase
`UPPER(`*`column`*`|`*`expression`*`)`	Converts alpha character values to uppercase
`INITCAP(`*`column`*`|`*`expression`*`)`	Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase. This function is only available in Oracle SQL.
`CONCAT(`*`column1`*`|`*`expression1`*`,` *`column2`*`|`*`expression2`*`)`	Concatenates the first character value to the second character value; equivalent to concatenation operator (||)
`SUBSTR(`*`column`*`|`*`expression`*`,`*`m`*`[,`*`n`*`])`	Returns specified characters from character value starting at character position *m*, *n* characters long (If *m* is negative, the count starts from the end of the character value. If *n* 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.

Oracle Internal & Oracle Academy Use Only

Function	Purpose
LENGTH(*column*|*expression*)	Returns the number of characters in the expression
INSTR(*column*|*expression*, *'string', [,m], [n]*)	Returns the numeric position of a named string. Optionally, you can provide a position *m* to start searching, and the occurrence *n* of the string. *m* and *n* default to 1, meaning start the search at the beginning of the string and report the first occurrence.
LPAD(*column*|*expression*, *n*, '*string*') RPAD(*column*|*expression*, *n*, '*string*')	Returns an expression left-padded to length of *n* characters with a character expression Returns an expression right-padded to length of *n* characters with a character expression
TRIM(*leading*|*trailing*|*both*, *trim_character FROM trim_source*)	Enables you to trim leading or trailing characters (or both) from a character string. If *trim_character* or *trim_source* is a character literal, you must enclose it in single quotation marks.
REPLACE(*text*, *search_string*, *replacement_string*)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Oracle Internal & Oracle Academy Use Only

Case-Conversion Functions

You can use the LOWER and UPPER functions to convert the case of character strings.

For example:

```
SELECT last_name, UPPER(last_name), job_id, LOWER(job_id)
FROM   employees
WHERE  department_id = 90;
```

	LAST_NAME	UPPER(LAST_NAME)	JOB_ID	LOWER(JOB_ID)
1	King	KING	AD_PRES	ad_pres
2	Kochhar	KOCHHAR	AD_VP	ad_vp
3	De Haan	DE HAAN	AD_VP	ad_vp

#	last_name	UPPER(last_name)	job_id	LOWER(job_id)
1	King	KING	AD_PRES	ad_pres
2	Kochhar	KOCHHAR	AD_VP	ad_vp
3	De Haan	DE HAAN	AD_VP	ad_vp

13

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: (Oracle) Converts the first letter of each word to uppercase and the remaining letters to lowercase

The slide shows an example of using LOWER and UPPER functions.

Oracle Internal & Oracle Academy Use Only

Using Case-Conversion Functions in WHERE Clauses in Oracle

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

14

The example in the slide 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.

Oracle Internal & Oracle Academy Use Only

Case-Insensitive Queries in MySQL

By default, MySQL uses a case-insensitive character set and collation. Sorting and string comparisons consider upper and lower case of the same character to be equivalent in value. To display the employee number, name, and department number for employee Higgins:

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

#	employee_ic	last_name	department_ic
1	205	Higgins	110
*	NULL	NULL	NULL

15

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

The `WHERE` clause of the SQL statement specifies the employee name as `higgins`. Even though all the data in the `employees` table is stored in proper case, the name `'higgins'` finds a match in the table and displays the results in the proper case. Any other combination, like `'hiGGins'`, `'HIGGINS'`, or 'Higgins' would also match.

The lesson titled "Restricting and Sorting Data" explained that, by default, MySQL uses a case-insensitive character set and collation, so that in sorting, upper and lower case letters sort as equivalent values. Also, in comparisons in a `WHERE` clause, upper and lower case letters are considered equivalent and are found as matches.

Oracle Internal & Oracle Academy Use Only

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*****`

16

`CONCAT`, `SUBSTR`, `LENGTH`, `INSTR`, `LPAD`, and `RPAD` are the character-manipulation functions that you will learn in this lesson.

- **`CONCAT`:** Joins values together (you are limited to using two parameters with `CONCAT`)
- **`SUBSTR`:** Extracts a string of determined length
- **`LENGTH`:** Shows the length of a string as a numeric value
- **`INSTR`:** Finds the numeric position of a named character
- **`LPAD`:** Returns an expression left-padded to the length of *n* characters with a character expression
- **`RPAD`:** Returns an expression right-padded to the length of *n* characters with a character expression

Oracle Internal & Oracle Academy Use Only

Using Character-Manipulation Functions

1

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

	LAST_NAME	JOB
1	Abel	Job category is SA_REP
2	Fay	Job category is MK_REP
3	Grant	Job category is SA_REP
4	Taylor	Job category is SA_REP

#	last_name	Job
1	Abel	Job category is SA_REP
2	Taylor	Job category is SA_REP
3	Grant	Job category is SA_REP
4	Fay	Job category is MK_REP

2

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

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

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

17

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.”

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

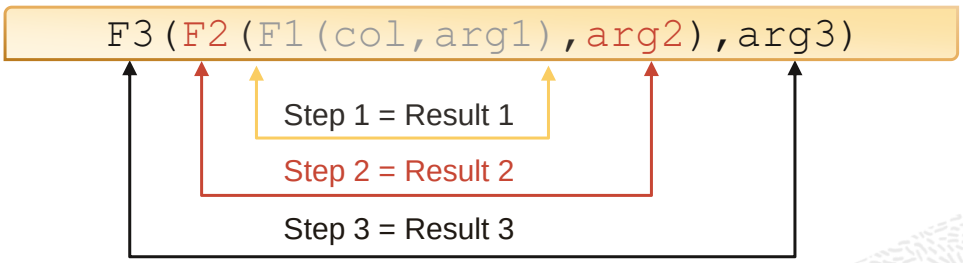
- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

18

Oracle Internal & Oracle Academy Use Only

Nesting Functions

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



19

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.

Oracle Internal & Oracle Academy Use Only

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

#	last_name	UPPER(CONCAT(SUBSTR(LAST_I
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

20

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

21

Oracle Internal & Oracle Academy Use Only

Numeric Functions

- ROUND: Rounds value to a specified decimal
- TRUNC (Oracle) or TRUNCATE (MySQL): 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
TRUNCATE(45.926, 2)	45.92
CEIL(2.83)	3
FLOOR(2.83)	2
MOD(1600, 300)	100

22

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

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

Note: This list contains only some of the available numeric functions. More information is available in the Oracle and MySQL documentation.

Oracle Internal & Oracle Academy Use Only

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

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

23

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 in Oracle

In Oracle, 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, in Oracle, both SELECT and FROM clauses are mandatory, and several calculations do not need to select from the actual tables.

MySQL does not create a DUAL table and does not require its use but accepts the FROM DUAL clause for compatibility.

Oracle Internal & Oracle Academy Use Only

Using the TRUNC Function in Oracle

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

	TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-1)
1	45.92	45	40

24

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.

Oracle Internal & Oracle Academy Use Only

Using the TRUNCATE Function in MySQL

```
SELECT TRUNCATE(45.923,2), TRUNCATE(45.923,0),
       TRUNCATE(45.923,-1);
```

#	TRUNCATE(45.923,2)	TRUNCATE(45.923,0)	TRUNCATE(45.923,-1)
1	45.92	45	40

25

The TRUNCATE function truncates the column, expression, or value to *n* decimal places.

The TRUNCATE function works with arguments similar to those of the ROUND function, except that **both** arguments are required.

If the second argument is 0, the value is truncated to zero decimal places.

If the second argument is a positive number, the value is truncated to that number of decimal places to the right of the decimal point (tenths, hundredths, thousandths etc.)

Conversely, if the second argument is a negative number, the value is truncated to that number of decimal places to the left of the decimal point (tens, hundreds, thousands, etc.)

Note that the FROM DUAL clause is not required for MySQL.

Oracle Internal & Oracle Academy Use Only

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

#	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

26

The `MOD` function finds the remainder of the first argument divided by the second argument. The example in the slide displays employee records where the `employee_id` is an even number.

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

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

27

Oracle Internal & Oracle Academy Use Only

Working with Dates in Oracle Databases

- 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`.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '01-FEB-2013';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-11
2	Kochhar	21-SEP-09
3	De Haan	13-JAN-09

...

28

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.
  - 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

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.

Oracle Internal & Oracle Academy Use Only

RR Date Format in Oracle

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.

29

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

Centuries and the Year 2000

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

Oracle Internal & Oracle Academy Use Only

Using the SYSDATE Function in Oracle

Use the SYSDATE function to get:

- Date
- Time

```
SELECT sysdate
FROM   dual;
```

	SYSDATE
1	28-JUN-18

30

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.

Oracle Internal & Oracle Academy Use Only

Using the CURRENT_DATE and CURRENT_TIMESTAMP Functions in Oracle

- CURRENT_DATE returns the current date from the user session.

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

	SESSIONTIMEZONE	CURRENT_DATE
1	UTC	28-JUN-18

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

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

	SESSIONTIMEZONE	CURRENT_TIMESTAMP
1	UTC	28-JUN-18 10.06.46.187191000 AM UTC

31

The CURRENT_DATE and CURRENT_TIMESTAMP functions return the current date and current time stamp, respectively.

Note: The SESSIONTIMEZONE function returns the value of the current session's time zone. The return type is a time zone offset (a character type in the format '[+|-]TZH:TZM') or a time zone region name, depending on how the user specified the session time zone value in the most recent ALTER SESSION statement. The example in the slide shows that the session time zone is Universal timezone. Observe that the database time zone is different from the current session's time zone.

Oracle Internal & Oracle Academy Use Only

Arithmetic with Dates in Oracle

- 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.

32

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

You can perform the following operations:

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

Oracle Internal & Oracle Academy Use Only

Using Arithmetic Operators with Dates in Oracle

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

	LAST_NAME	WEEKS
1	King	478.8719179894179894179894179894179894 18
2	Kochhar	360.7290608465608465608465608465608465 61
3	De Haan	605.3004894179894179894179894179894179 89

33

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

34

Oracle Internal & Oracle Academy Use Only

Working with Dates in MySQL Databases

The MySQL default date entry and display format is 'YYYY-MM-DD'. To display employees hired before February 1, 2013, enter the following query:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date < '2013-02-01';
```

#	last_name	hire_date
1	King	2011-06-17
2	Kochhar	2009-09-21
3	De Haan	2009-01-13
4	Rajs	2011-10-17
5	Davies	2013-01-29
6	Abel	2012-05-11
7	Whalen	2011-09-17
8	Hartstein	2012-02-17
9	Higgins	2010-06-07
10	Gietz	2010-06-07

...

35

The supported range of dates is `'100-01-01'` to `'9999-12-31'`. The `MM` portion of the date must be between 1 and 12 and the `DD` portion between 1 and 31. MySQL accepts dates in several formats, including other delimiters or no delimiters. For example, any of the following would be accepted for February 1, 2013:

`'2013-02-01'`

`'2013/02/01'`

`'20130201'`

Oracle Internal & Oracle Academy Use Only

Displaying the Current Date in MySQL

The `CURDATE()` function returns the current date.

- `CURRENT_DATE()` and `CURRENT_DATE` are synonyms for `CURDATE()`.

The `NOW()` function returns the current date and time.

- `CURRENT_TIMESTAMP()` and `CURRENT_TIMESTAMP` are synonyms for `NOW()`.

The `SYSDATE()` function returns the current date and time.

```
SELECT CURDATE(), NOW(), SYSDATE();
```

#	CURDATE()	NOW()	SYSDATE()
1	2018-08-22	2018-08-22 15:33:14	2018-08-22 15:33:14

36

You can use the `CURDATE()` function in any statement that requires a date and `NOW()` in any statement that requires a date and time combination.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates in Oracle Databases
- Working with dates in MySQL Databases
- Date functions

37

Oracle Internal & Oracle Academy Use Only

Date-Manipulation Functions in Oracle

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

38

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."

Oracle Internal & Oracle Academy Use Only

Using Date Functions in Oracle

Function	Result
MONTHS_BETWEEN ('01-SEP-18','11-JAN-17')	19.6774194
ADD_MONTHS ('31-JAN-16',1)	'29-FEB-16'
NEXT_DAY ('01-JUN-16','FRIDAY')	'08-JUN-18'
LAST_DAY ('01-APR-16')	'30-APR-18'

39

In the example in the slide, the `ADD_MONTHS` function adds one month to the supplied date value "31-JAN-16" and returns "29-FEB-16." The function recognizes the year 2016 as a leap year and, therefore, returns the last day of the February month. If you change the input date value to "31-JAN-18," the function returns "28-FEB-18."

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

```
SELECT employee_id, hire_date, MONTHS_BETWEEN (SYSDATE, hire_date)
   TENURE, ADD_MONTHS (hire_date, 6) REVIEW,  NEXT_DAY (hire_date,
   'FRIDAY'), LAST_DAY(hire_date)
FROM employees WHERE  MONTHS_BETWEEN (SYSDATE, hire_date) < 150;
```

Oracle Internal & Oracle Academy Use Only

Using ROUND and TRUNC Functions with Dates in Oracle

Assumption: The below functions were run on 29-JUN-18.

Function	Result
ROUND(SYSDATE,'MONTH')	01-JUL-18
ROUND(SYSDATE,'YEAR')	01-JAN-18
TRUNC(SYSDATE,'MONTH')	01-JUN-18
TRUNC(SYSDATE,'YEAR')	01-JAN-18

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';
```

Oracle Internal & Oracle Academy Use Only

Date-Manipulation Functions in MySQL

Function	Result
`DATE_ADD(date, INTERVAL expr unit)`	Date after an interval is added to a date
`DATE_SUB(date, INTERVAL expr unit)`	Date after an interval is subtracted from a date
`DATEDIFF(date, date)`	Difference in days between two dates
`LAST_DAY(date)`	Last day of the month
`MONTH(date)`	The month number of the date
`YEAR(date)`	The year of the date

41

Date functions operate on MySQL dates. The functions listed in the slide are just a small portion of the functions available to operate on MySQL dates.

Oracle Internal & Oracle Academy Use Only

Using Date Functions in MySQL

The following example uses MySQL date functions in the output as well as in the `WHERE` clause:

```
SELECT employee_id, hire_date,
DATE_ADD(hire_date, INTERVAL 6 MONTH) AS Review,
DATEDIFF(CURDATE(), hire_date) AS Tenure
FROM employees
WHERE hire_date > DATE_SUB(CURDATE(), INTERVAL 4 YEAR);
```

#	employee_ic	hire_date	Review	Tenure
1	104	2015-05-21	2015-11-21	1189
2	107	2015-02-07	2015-08-07	1292
3	124	2015-11-16	2016-05-16	1010
4	149	2016-01-29	2016-07-29	936
5	178	2015-05-24	2015-11-24	1186

The query displays the following:

- Employee ID
- Hire date
- The date of the employee's first six-month review with a column heading of Review. This is calculated by adding an interval of 6 months to the hire date.
- The number of days since the employee was hired with a column heading of Tenure. This is calculated as the difference between the hire date and the current date.

The query selects those employees who were hired less than 4 years ago. It compares the hire date to see if it is later (greater) than the date that was four years before today, which is calculated by subtracting an interval of 4 years from the current date.

Oracle Internal & Oracle Academy Use Only

Extracting the Month or Year Portion of Dates in MySQL

Use the `MONTH()` or `YEAR()` function to extract those portions of a date. For example, the following query displays the employee number, hire date, and starting month for employees that started in `2010`.

```
SELECT employee_id, hire_date,
MONTH(hire_date)
FROM employees
WHERE YEAR(hire_date) = '2010';
```

#	employee_id	hire_date	MONTH(hire_date)
1	205	2010-06-07	6
2	206	2010-06-07	6

43

There are a great many other functions that enable you to work with dates in MySQL

Oracle Internal & Oracle Academy Use Only

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 Internal & Oracle Academy Use Only

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

45

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

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

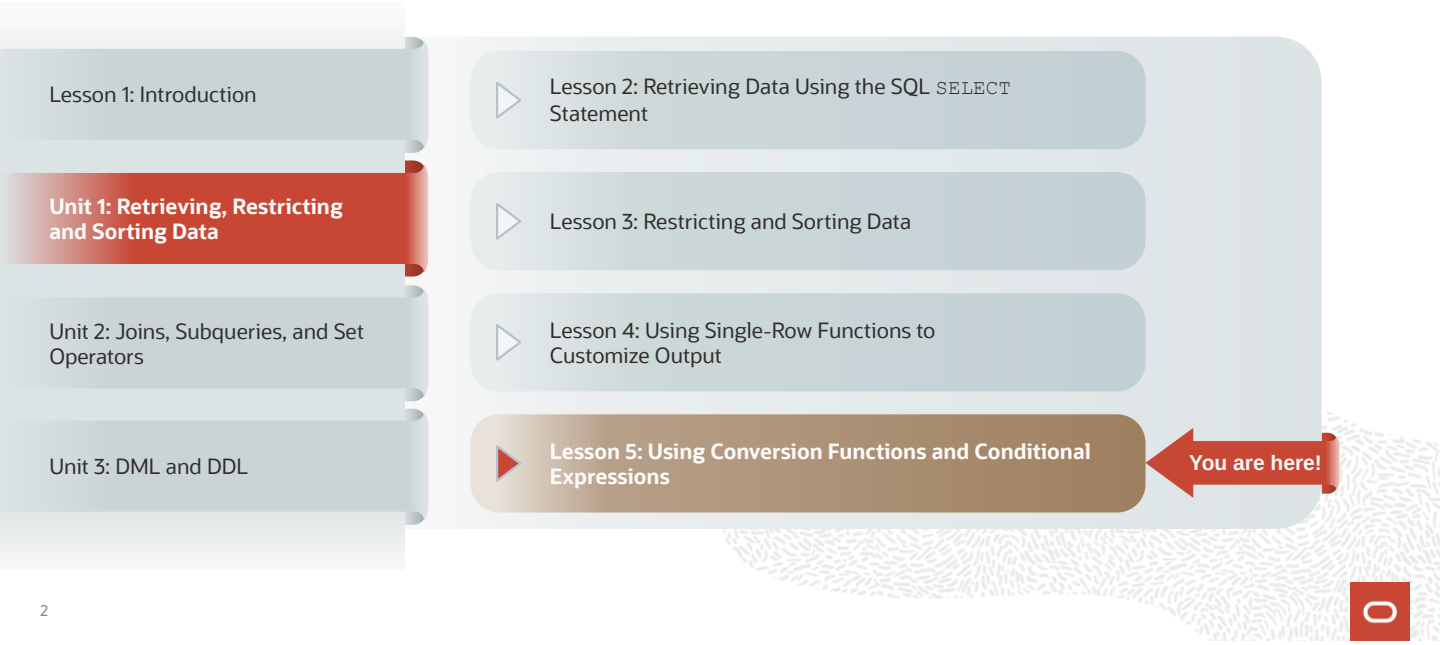
ORACLE

5

Using Conversion Functions and Conditional Expressions

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 1, you will learn how to query the data from tables, how to query selected records from tables, and also how to sort the data retrieved from the tables.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- 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

3

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

Oracle Internal & Oracle Academy Use Only

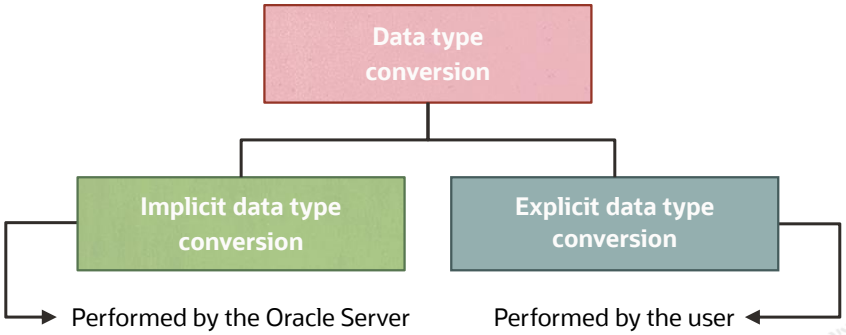
Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions in Oracle
- Using the CAST() function in MySQL
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - Searched `CASE`
  - `DECODE`
- JSON functions:
  - `JSON_QUERY`
  - `JSON_TABLE`
  - `JSON_VALUE`

4

Oracle Internal & Oracle Academy Use Only

Conversion Functions



5

In some situations, the database server is provided data of one type when it expects data of a different data type. This can happen when providing output, when making comparisons in a WHERE clause, or when using a function that expects a specific data type. When this happens, the database 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.

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.

Oracle Internal & Oracle Academy Use Only

Implicit Data Type Conversion of Strings to Numbers

In expressions, the database can automatically convert strings to numbers. In this example, two strings are concatenated and then implicitly converted to a number for comparison with the numeric department ID in the `WHERE` clause.

```
SELECT first_name, last_name, department_id
FROM employees WHERE department_id < CONCAT('9', '0');
```

	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
1	Ellen	Abel	80
2	Curtis	Davies	50
3	Bruce	Ernst	60
4	Pat	Fay	20
5	Michael	Hartstein	20
6	Alexander	Hunold	60
7	Diana	Lorentz	60
8	Randall	Matos	50
9	Kevin	Mourgos	50
10	Trenna	Rajs	50
11	Jonathon	Taylor	80
12	Peter	Vargas	50
13	Jennifer	Whalen	10
14	Eleni	Zlotkey	80

#	first_name	last_name	department_id
1	Alexander	Hunold	60
2	Bruce	Ernst	60
3	Diana	Lorentz	60
4	Kevin	Mourgos	50
5	Trenna	Rajs	50
6	Curtis	Davies	50
7	Randall	Matos	50
8	Peter	Vargas	50
9	Eleni	Zlotkey	80
10	Ellen	Abel	80
11	Jonathon	Taylor	80
12	Jennifer	Whalen	10
13	Michael	Hartstein	20
14	Pat	Fay	20

6

The server can automatically perform data type conversion in an expression from `VARCHAR2` (Oracle), `VARCHAR` (MySQL) or `CHAR` to a numeric or date data type.

For example, the expression `hire_date > '01-JAN-90'` (Oracle) or `hire_date > '1990-01-01'` (MySQL) results in the implicit conversion from the string to a date. Therefore, a character expression or value can be implicitly converted to a number or date data type in an expression.

Note: `CHAR` to numeric data type conversions succeed only if the character string provided represents a valid number.

Oracle Internal & Oracle Academy Use Only

Implicit Data Type Conversion of Numbers to Strings

In expressions, the database can automatically convert numbers to strings. In this example, the salary column is converted to a string to determine if it contains a character.

```
SELECT first_name, last_name, salary
FROM employees
WHERE INSTR(salary, '5') > 0;
```

	FIRST_NAME	LAST_NAME	SALARY
1	Kevin	Mourgos	5800
2	Trenna	Rajs	3500
3	Peter	Vargas	2500
4	Eleni	Zlotkey	10500

#	first_name	last_name	salary
1	Kevin	Mourgos	5800.00
2	Trenna	Rajs	3500.00
3	Peter	Vargas	2500.00
4	Eleni	Zlotkey	10500.00

7

The Oracle server can automatically perform data type conversion in an expression from `NUMBER` or `DATE` to `VARCHAR2` or `CHAR`.

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

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions in Oracle
- Using the `CAST()` function in MySQL
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - Searched `CASE`
  - `DECODE`
- JSON functions:
  - `JSON_QUERY`
  - `JSON_TABLE`
  - `JSON_VALUE`

8

Oracle Internal & Oracle Academy Use Only

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

9

You can convert a datetime data type to a value of `VARCHAR2` data type using `TO_CHAR` in Oracle 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.

Oracle Internal & Oracle Academy Use Only

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

10

You can refer to the complete list of date format model elements in the following page.

Oracle Internal & Oracle Academy Use Only

Element	Description
SCC or CC	Century; server prefixes B.C. date with -
Years in dates YYYY or SYYYY	Year; server prefixes B.C. date with -
YYY or YY or Y	Last three, two, or one digit of the year
Y,YYY	Year with comma in this position
IYYY, IYY, IY, I	Four-, three-, two-, or one-digit year based on the ISO standard
SYEAR or YEAR	Year spelled out; server prefixes B.C. date with -
BC or AD	Indicates B.C. or A.D. year
B.C. or A.D.	Indicates B.C. or A.D. year using periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of the month padded with blanks to a length of nine characters
MON	Name of the month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of the year or month
DDD or DD or D	Day of the year, month, or week
DAY	Name of the day padded with blanks to a length of nine characters
DY	Name of the day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.
IW	Weeks in the year from ISO standard (1 to 53)

Oracle Internal & Oracle Academy Use Only

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`

12

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)
SSSSS	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)

Oracle Internal & Oracle Academy Use Only

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

...

13

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')
 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.

Oracle Internal & Oracle Academy Use Only

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

14

When working with number values, such as character strings, you should convert the numbers to the character data type using the TO_CHAR function. This function translates a value of NUMBER data type to VARCHAR2 data type.

This technique is especially useful with concatenation.

Oracle Internal & Oracle Academy Use Only

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

Oracle Internal & Oracle Academy Use Only

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

16

Good to know:

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

Oracle Internal & Oracle Academy Use Only

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'])
```

17

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

The fx modifier in these functions specifies the exact match for the character argument and date format model of a TO_DATE function:

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

Oracle Internal & Oracle Academy Use Only

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');
```

Oracle Internal & Oracle Academy Use Only

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')
1	Kochhar	21-Sep-2009
2	De Haan	13-Jan-2009

19

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions in Oracle
- Using the `CAST()` function
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - Searched `CASE`
  - `DECODE`
- JSON functions:
  - `JSON_QUERY`
  - `JSON_TABLE`
  - `JSON_VALUE`

20

Oracle Internal & Oracle Academy Use Only

Using the CAST() function in Oracle

CAST lets you convert one data type to another.

```
CAST(input_value as destination_type)
```

Examples:

1

```
SELECT first_name, last_name, department_id
FROM employees
WHERE department_id < CAST(CONCAT('9', '0') AS
DECIMAL(2,0));
```

2

```
SELECT first_name, last_name, salary
FROM employees
WHERE INSTR(CAST(salary AS VARCHAR2(30)), '5')
> 0;
```

21

CAST lets you convert built-in data types of one type into another built-in data type. You can convert a string to a number or a date, a number to a string, etc.

In the first example, two strings are concatenated and then explicitly converted to a DECIMAL numeric data type using the CAST() function to compare to the numeric department ID. The query displays the first name, last name, department ID of those employees whose department ID is less than 90.

In the second example, the salary column is explicitly converted to a string using the CAST() function to determine if it contains a character. The query display the first name, last name, and salary of employees whose salary contains the digit '5'.

For more information on the CAST() function, refer to the *Oracle Database 18c SQL Language Reference* guide.

Oracle Internal & Oracle Academy Use Only

Explicit Data Type Conversion of Strings to Numbers in MySQL

You can use the `CAST()` function to explicitly convert strings to numbers. In this example, two strings are concatenated and then explicitly converted to a `DECIMAL` numeric data type to compare to the numeric department ID.

```
SELECT first_name, last_name, department_id FROM employees
WHERE department_id < CAST(CONCAT('9', '0') AS DECIMAL(2,0));
```

#	first_name	last_name	department_id
1	Alexander	Hunold	60
2	Bruce	Ernst	60
3	Diana	Lorentz	60
4	Kevin	Mourgos	50
5	Trenna	Rajs	50
6	Curtis	Davies	50
7	Randall	Matos	50
8	Peter	Vargas	50
9	Eleni	Zlotkey	80
10	Ellen	Abel	80
11	Jonathon	Taylor	80
12	Jennifer	Whalen	10
13	Michael	Hartstein	20
14	Pat	Fay	20

22

The `CAST()` function explicitly converts from a character to a decimal number value. (`INTEGER` data type is not accepted.)

Oracle Internal & Oracle Academy Use Only

Explicit Data Type Conversion of Numbers to Strings in MySQL

You can use the `CAST()` function to explicitly convert strings to numbers. In this example, the salary column is explicitly converted to a string to determine if it contains a character.

```
SELECT first_name, last_name, salary
FROM employees
WHERE INSTR(CAST(salary AS CHAR), '5');
```

#	first_name	last_name	salary
1	Kevin	Mourgos	5800.00
2	Trenna	Rajs	3500.00
3	Peter	Vargas	2500.00
4	Eleni	Zlotkey	10500.00

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions in Oracle
- Using the `CAST()` function in MySQL
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - Searched `CASE`
  - `DECODE`
- JSON functions:
  - `JSON_QUERY`
  - `JSON_TABLE`
  - `JSON_VALUE`

24

Oracle Internal & Oracle Academy Use Only

General Functions

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

```
NVL (expr1, expr2)
```

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

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

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

```
IFNULL (expr1, expr2)
In MySQL
```

25

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.
IFNULL	In MySQL, IFNULL substitutes a NULL value with a value that can be used in an expression. If *expr1* is NULL, returns *expr2*, otherwise returns *expr1*.
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 19c database.

Oracle Internal & Oracle Academy Use Only

NVL Function (Oracle) and IFNULL () Function (MySQL)

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types for both expressions must match.
- Examples:

Oracle	MySQL
NVL(commission_pct,0)	IFNULL(commission_pct,0)
NVL(hire_date,'01-JAN-97')	IFNULL(hire_date, '1997-01-01')
NVL(job_id,'No Job Yet')	IFNULL(job_id,'No Job Yet')

NVL (expr1, expr2)

IFNULL(expr1, expr2)

26

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

Syntax

NVL (*expr1*, *expr2*)

IFNULL (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	NVL(*number_column*,9)
DATE	NVL(*date_column*, '01-JAN-95')
CHAR or VARCHAR2	NVL(*character_column*, 'Unavailable')

Oracle Internal & Oracle Academy Use Only

Using the NVL Function in Oracle

```
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

27

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.

Oracle Internal & Oracle Academy Use Only

Using the NVL2 Function in Oracle

```
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

28

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.

Oracle Internal & Oracle Academy Use Only

Using the IFNULL Function in MySQL

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

#	last_name	salary	IFNULL(commission_pct,	AN_SAL
1	King	24000.00	0.00	288000.0000
2	Kochhar	17000.00	0.00	204000.0000
3	De Haan	17000.00	0.00	204000.0000
...				
12	Zlotkey	10500.00	0.20	151200.0000
13	Abel	11000.00	0.30	171600.0000
14	Taylor	8600.00	0.20	123840.0000
...				
18	Fay	6000.00	0.00	72000.0000
19	Higgins	12008.00	0.00	144096.0000
20	Gietz	8300.00	0.00	99600.0000

29

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.

Oracle Internal & Oracle Academy Use Only

Using the NULLIF Function

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

```
SELECT first_name, LENGTH(first_name) AS expr1,   (1)
       last_name,  LENGTH(last_name)  AS expr2,   (2)
       NULLIF(LENGTH(first_name), LENGTH(last_name)) AS Result   (3)
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)

...

first_name	expr1	last_name	expr2	Result
Ellen	5	Abel	4	5
Curtis	6	Davies	6	NULL
Lex	3	De Haan	7	3
Bruce	5	Ernst	5	NULL
Pat	3	Fay	3	NULL
William	7	Gietz	5	7
Kimberely	9	Grant	5	9
Michael	7	Hartstein	9	7
Shelley	7	Higgins	7	NULL

...

30

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.

Oracle Internal & Oracle Academy Use Only

Using the COALESCE Function

- The advantage of the COALESCE function over the NVL or IFNULL functions 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)
```

31

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.

Oracle Internal & Oracle Academy Use Only

Using the COALESCE Function

```
SELECT last name, salary, commission pct,
COALESCE((salary+(commission_pct*salary)), salary+2000)
AS New_Salary
FROM   employees;
```

	LAST_NAME	SALARY	COMMISSION_PCT	NEW_SALARY
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

#	last_name	salary	commission_p	New_Salary
1	King	24000.00	NULL	26000.0000
2	Kochhar	17000.00	NULL	19000.0000
3	De Haan	17000.00	NULL	19000.0000
4	Hunold	9000.00	NULL	11000.0000
5	Ernst	6000.00	NULL	8000.0000
6	Lorentz	4200.00	NULL	6200.0000
7	Mourgos	5800.00	NULL	7800.0000
8	Rajs	3500.00	NULL	5500.0000
9	Davies	3100.00	NULL	5100.0000
10	Matos	2600.00	NULL	4600.0000
11	Vargas	2500.00	NULL	4500.0000
12	Zlotkey	10500.00	0.20	12600.0000
13	Abel	11000.00	0.30	14300.0000
14	Taylor	8600.00	0.20	10320.0000
15	Grant	7000.00	0.15	8050.0000
16	Whalen	4400.00	NULL	6400.0000
17	Hartstein	13000.00	NULL	15000.0000
18	Fay	6000.00	NULL	8000.0000
19	Higgins	12008.00	NULL	14008.0000
20	Gietz	8300.00	NULL	10300.0000

32

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions in Oracle
- Using the `CAST()` function in MySQL
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - Searched `CASE`
  - `DECODE`
- JSON functions:
  - `JSON_QUERY`
  - `JSON_TABLE`
  - `JSON_VALUE`

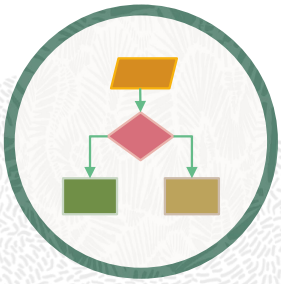
33

Oracle Internal & Oracle Academy Use Only

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



34

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.

Oracle Internal & Oracle Academy Use Only

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
```

35

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.

Oracle Internal & Oracle Academy Use Only

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     AS REVISED_SALARY
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
1	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

#	last_name	job_id	salary	REVISED_SALARY
1	King	AD_PRES	24000.00	24000.00
...				
4	Hunold	IT_PROG	9000.00	9900.0000
5	Ernst	IT_PROG	6000.00	6600.0000
6	Lorentz	IT_PROG	4200.00	4620.0000
7	Mourgos	ST_MAN	5800.00	5800.00
8	Rajs	ST_CLERK	3500.00	4025.0000
9	Davies	ST_CLERK	3100.00	3565.0000
10	Matos	ST_CLERK	2600.00	2990.0000
11	Vargas	ST_CLERK	2500.00	2875.0000
...				
13	Abel	SA_REP	11000.00	13200.0000
14	Taylor	SA_REP	8600.00	10320.0000
15	Grant	SA_REP	7000.00	8400.0000

36

In the SQL statement in the slide, the value of `JOB_ID` is decoded.

If `JOB_ID` is `IT_PROG`, the salary increase is 10%.

If `JOB_ID` is `ST_CLERK`, the salary increase is 15%.

If `JOB_ID` is `SA_REP`, the salary increase is 20%.

For all other job roles, there is no increase in salary.

The same statement can be written with the `DECODE` function.

Oracle Internal & Oracle Academy Use Only

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) AS qualified_salary
FROM employees;
```

37

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 all must have a numeric data type.

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

Oracle Internal & Oracle Academy Use Only

DECODE Function in Oracle

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])
```

38

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

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

Oracle Internal & Oracle Academy Use Only

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	Rajs	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

39

In the SQL statement in the slide, the value of JOB_ID is tested.

If JOB_ID is IT_PROG, the salary increase is 10%.

If JOB_ID is ST_CLERK, the salary increase is 15%.

If JOB_ID is SA_REP, the salary increase is 20%.

For all other job roles, there is no increase in salary.

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

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

Oracle Internal & Oracle Academy Use Only

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;
```

40

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%

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions in Oracle
- Using the `CAST()` function in MySQL
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - CASE
  - Searched CASE
  - DECODE
- JSON functions:
  - `JSON_QUERY`
  - `JSON_TABLE`
  - `JSON_VALUE`

41

Oracle Internal & Oracle Academy Use Only

JSON_QUERY Function

The SQL/JSON function JSON_QUERY finds one or more specified JSON values in JSON data and returns the values in a character string.

```
SELECT JSON_QUERY('{a:100, b:200, c:300}', '$') AS value
  FROM DUAL;

VALUE
--------------------------------------------------------------------------------
{"a":100,"b":200,"c":300}
```

42

The following query returns the context item or the specified string of JSON data. The path expression matches a single JSON object, which does not require an array wrapper. Note that the JSON data is converted to strict JSON syntax in the returned value—that is, the object property names are enclosed in double quotation marks.

Note: See Appendix C in *Oracle Database Globalization Support Guide* for the collation derivation rules, which define the collation assigned to the character value returned by JSON_QUERY.

Oracle Internal & Oracle Academy Use Only

JSON_TABLE Function

The SQL/JSON function JSON_TABLE creates a relational view of JSON data.

```
SELECT JSON_QUERY('{a:100, b:200, c:300}', '$') AS value
  FROM DUAL;

VALUE
--------------------------------------------------------------------------------
{"a":100,"b":200,"c":300}
```

43

It maps the result of a JSON data evaluation into relational rows and columns. You can query the result returned by the function as a virtual relational table using SQL. The main purpose of JSON_TABLE is to create a row of relational data for each object inside a JSON array and output JSON values from within that object as individual SQL column values.

You must specify JSON_TABLE only in the FROM clause of a SELECT statement. The function first applies a path expression, called a **SQL/JSON row path expression**, to the supplied JSON data. The JSON value that matches the row path expression is called a **row source** in that it generates a row of relational data. The COLUMNS clause evaluates the row source, finds specific JSON values within the row source, and returns those JSON values as SQL values in individual columns of a row of relational data.

Note: See Appendix C in *Oracle Database Globalization Support Guide* for the collation derivation rules, which define the collation assigned to each character data type column in the table generated by JSON_TABLE.

Oracle Internal & Oracle Academy Use Only

JSON_VALUE Function

The SQL/JSON function JSON_QUERY finds one or more specified JSON values in JSON data and returns the values in a character string.

```
SELECT JSON_VALUE('{a:100}', '$.a') AS value
  FROM DUAL;

VALUE
-----
100
```

44

The example query listed returns the value of the member with a property name. Because the RETURNING clause is not specified, the value is returned as a VARCHAR2(4000) data type.

Note: See Appendix C in *Oracle Database Globalization Support Guide* for the collation derivation rules, which define the collation assigned to the value returned by this function when it is a character value.

Oracle Internal & Oracle Academy Use Only

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

45

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.

Oracle Internal & Oracle Academy Use Only

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`

46

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.

Oracle Internal & Oracle Academy Use Only

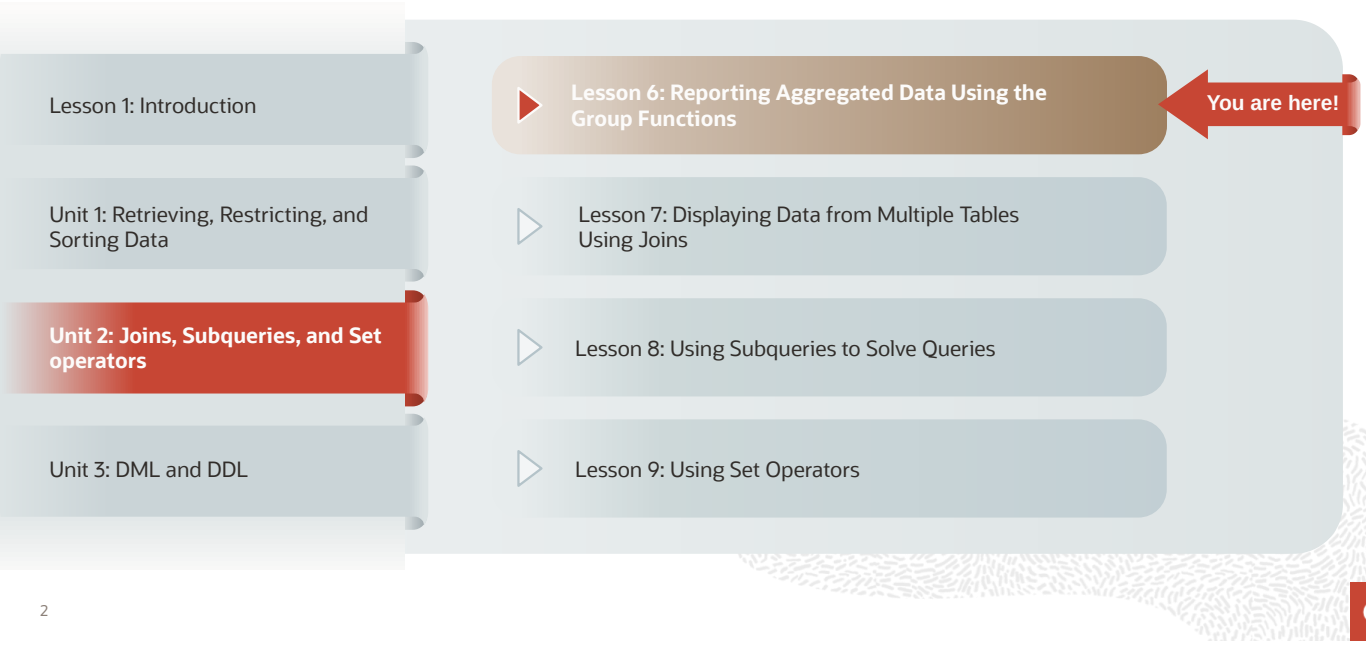
ORACLE

6

Reporting Aggregated Data Using the Group Functions

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 2, you will learn about SQL statements to query and display data from multiple tables using Joins. You will also learn to use subqueries when the condition is unknown, use group functions to aggregate data, and use set operators.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- 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

3

In this lesson, you will further learn about functions. The lesson focuses on obtaining summary information (such as averages) for groups of rows. You will discuss how to group rows in a table into smaller sets and how to specify search criteria for groups of rows.

Oracle Internal & Oracle Academy Use Only

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

4

Oracle Internal & Oracle Academy Use Only

Group Functions

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

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

5

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.

Oracle Internal & Oracle Academy Use Only

Types of Group Functions

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

6

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]*n*)	Average value of *n*, ignoring null values
COUNT([DISTINCT|ALL]expr)	Number of rows where *expr* evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX([DISTINCT|ALL]expr)	Maximum value of *expr*, ignoring null values
MIN([DISTINCT|ALL]*expr*)	Minimum value of *expr*, ignoring null values
STDDEV([DISTINCT|ALL]*n*)	Standard deviation of *n*, ignoring null values
SUM([DISTINCT|ALL]*n*)	Sum values of *n*, 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]*n*)	Variance of *n*, ignoring null values

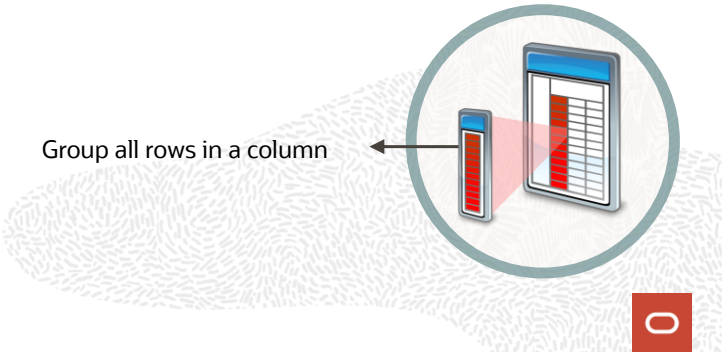
***Note:** Materialized views based on COUNT(DISTINCT) functions can provide enhanced performance by using bitmap-based operations on integer columns.

Starting with Oracle Database Release 19c, you can create materialized views based on SQL aggregate functions that use bitmap representation to express the computation of COUNT(DISTINCT) operations. These functions include
BITMAP_BUCKET_NUMBER, BITMAP_BIT_POSITION and BITMAP_CONSTRUCT_AGG.

Oracle Internal & Oracle Academy Use Only

Group Functions: Syntax

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



7

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.

Oracle Internal & Oracle Academy Use Only

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

#	AVG(salary)	MAX(salary)	MIN(salary)	SUM(salary)
1	8150.000000	11000.00	6000.00	32600.00

8

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.

Oracle Internal & Oracle Academy Use Only

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

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

	MIN(LAST_NAME)	MAX(LAST_NAME)	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	Abel	Zlotkey	13-JAN-09	29-JAN-16

#	MIN(last_name)	MAX(last_name)	MIN(hire_date)	MAX(hire_date)
1	Abel	Zlotkey	2009-01-13	2016-01-29

9

You can use the MAX and MIN functions for numeric, character, and date data types. The example in the slide displays the employee last name that is first and last in an alphabetic list of all employees and the date hired for the most senior and most junior 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.

Oracle Internal & Oracle Academy Use Only

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(*)
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

#	COUNT(commission_pc
1	0

10

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

1. The first example in the slide displays the number of employees in department 50.
2. The second example in the slide displays the number of employees in department 50 who can earn a commission.

Oracle Internal & Oracle Academy Use Only

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

#	COUNT(DISTINCT department_id
1	7

11

Use the DISTINCT keyword to suppress the counting of any duplicate values in a column.

The example in the slide displays the number of distinct department values that are in the EMPLOYEES table.

Oracle Internal & Oracle Academy Use Only

Group Functions and Null Values in Oracle

- Group functions ignore null values in the column:

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

	AVG(COMMISSION_PCT)
1	0.2125

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

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

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

12

All group functions ignore null values in the column.

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

Examples

1. The average is calculated based on *only* those rows in the table in which a valid value is stored in the COMMISSION_PCT column. The average is calculated as the total commission that is paid to all employees divided by the number of employees receiving a commission (four).
2. The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the COMMISSION_PCT column. The average is calculated as the total commission that is paid to all employees divided by the total number of employees in the company (20).

Oracle Internal & Oracle Academy Use Only

Group Functions and Null Values in MySQL

- Group functions ignore null values in the column:

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

#	avg(commission_pc
1	0.212500

- The `IFNULL` function forces group functions to include null values:

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

#	AVG(IFNULL(commission_pct, 0
1	0.042500

13

All group functions ignore null values in the column.

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

Examples

1. The average is calculated based on *only* those rows in the table in which a valid value is stored in the `COMMISSION_PCT` column. The average is calculated as the total commission that is paid to all employees divided by the number of employees receiving a commission (four).
2. The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the `COMMISSION_PCT` column. The average is calculated as the total commission that is paid to all employees divided by the total number of employees in the company (20).

Oracle Internal & Oracle Academy Use Only

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

14

Oracle Internal & Oracle Academy Use Only

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	2500	
5	50	2600	
6	50	3100	3500
7	50	3500	
8	50	5800	
9	60	9000	
10	60	6000	6400
11	60	4200	
12	80	11000	10033
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)	7000
2	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

15

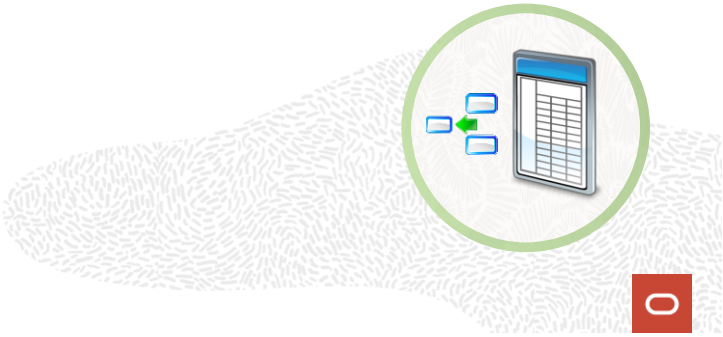
Until this point in the discussion, you have observed that all group functions have treated the table as one large group of information. At times, however, you need to divide the table of information into smaller groups. This can be done by using the `GROUP BY` clause.

Oracle Internal & Oracle Academy Use Only

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];
```



16

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.

Oracle Internal & Oracle Academy Use Only

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	19333.333333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.333333333333333333333333333333333
7	60	6400
8	10	4400

#	department_i	AVG(salary)
1	NULL	7000.000000
2	10	4400.000000
3	20	9500.000000
4	50	3500.000000
5	60	6400.000000
6	80	10033.333333
7	90	19333.333333
8	110	10154.000000

17

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.

Oracle Internal & Oracle Academy Use Only

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	19333.333333333333333333333333333333333
3	9500
4	10154
5	3500
6	10033.333333333333333333333333333333333
7	6400
8	4400

#	AVG(salary)
1	7000.000000
2	4400.000000
3	9500.000000
4	3500.000000
5	6400.000000
6	10033.333333
7	19333.333333
8	10154.000000

18

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);
```

Oracle Internal & Oracle Academy Use Only

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

19

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;
```

Oracle Internal & Oracle Academy Use Only

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

#	department_i(	job_id	SUM(salary)
1	50	ST_CLERK	11700.00
2	50	ST_MAN	5800.00
3	60	IT_PROG	19200.00
4	80	SA_MAN	10500.00
5	80	SA_REP	19600.00
6	90	AD_PRES	24000.00
7	90	AD_VP	34000.00
8	110	AC_ACC...	8300.00
9	110	AC_MGR	12008.00

20

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.

Oracle Internal & Oracle Academy Use Only

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

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

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

```
Error Code: 1140. In aggregated query without GROUP BY, expression #1 of SELECT list contains
nonaggregated column 'hr.employees.department_id'; this is incompatible with sql_mode=only_full_group_by
```

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

2

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

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

```
Error Code: 1055. Expression #2 of SELECT list is not in GROUP BY clause and contains nonaggregated
column 'hr.employees.job_id' which is not functionally dependent on columns in GROUP BY clause; this is
incompatible with sql_mode=only_full_group_by
```

21

1. 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;
```

2. 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;
```

Oracle Internal & Oracle Academy Use Only

Illegal Queries Using Group Functions in a WHERE Clause

- You use the HAVING clause to restrict groups using a group function.
- You cannot use group functions in the WHERE clause.

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

Cannot use a group function in a WHERE clause

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

Error Code: 1111. Invalid use of group function

22

You cannot use a group function in a WHERE clause. The SELECT statement in the example in the slide results in an error because it uses the AVG() function in 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 putting the group function in the HAVING clause to restrict groups:

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

Oracle Internal & Oracle Academy Use Only

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

23

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.

Oracle Internal & Oracle Academy Use Only

Restricting Group Results with the HAVING Clause

When you use the HAVING clause, the Oracle server restricts groups:

- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

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

24

You use the HAVING clause to specify the groups that are to be displayed, thus further restricting the groups on the basis of aggregate information.

The Oracle server performs the following steps when you use the HAVING clause:

1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the HAVING clause are displayed.

In the syntax, *group_condition* restricts the groups of rows returned to those groups for which the specified condition is true.

The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

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

Oracle Internal & Oracle Academy Use Only

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

#	department_i	MAX(salary)
1	20	13000.00
2	80	11000.00
3	90	24000.00
4	110	12008.00

25

The example in the slide displays the department numbers and maximum salaries for departments with a maximum salary greater than $10,000.

You can use the GROUP BY clause without using a group function in the SELECT list. If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.

The following example displays the department numbers and average salaries for departments with a maximum salary greater than $10,000:

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

Oracle Internal & Oracle Academy Use Only

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

#	job_id	PAYROLL
1	IT_PROG	19200.00
2	AD_PRES	24000.00
3	AD_VP	34000.00

26

The example in the slide displays the JOB_ID and total monthly salary for each job that has a total payroll exceeding $13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

Oracle Internal & Oracle Academy Use Only

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

27

Oracle Internal & Oracle Academy Use Only

Nesting Group Functions in Oracle

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

	MAX(AVG(SALARY))
1	19333.3333333333333333333333333333333333

28

Group functions can be nested to a depth of two functions in Oracle SQL. 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.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Use the group functions `COUNT, MAX, MIN, SUM,` and `AVG`
- Write queries that use the `GROUP BY` clause
- Write queries that use the `HAVING` clause

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

29

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 19c database.

Oracle Internal & Oracle Academy Use Only

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

30

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 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 the 19c database.

Oracle Internal & Oracle Academy Use Only

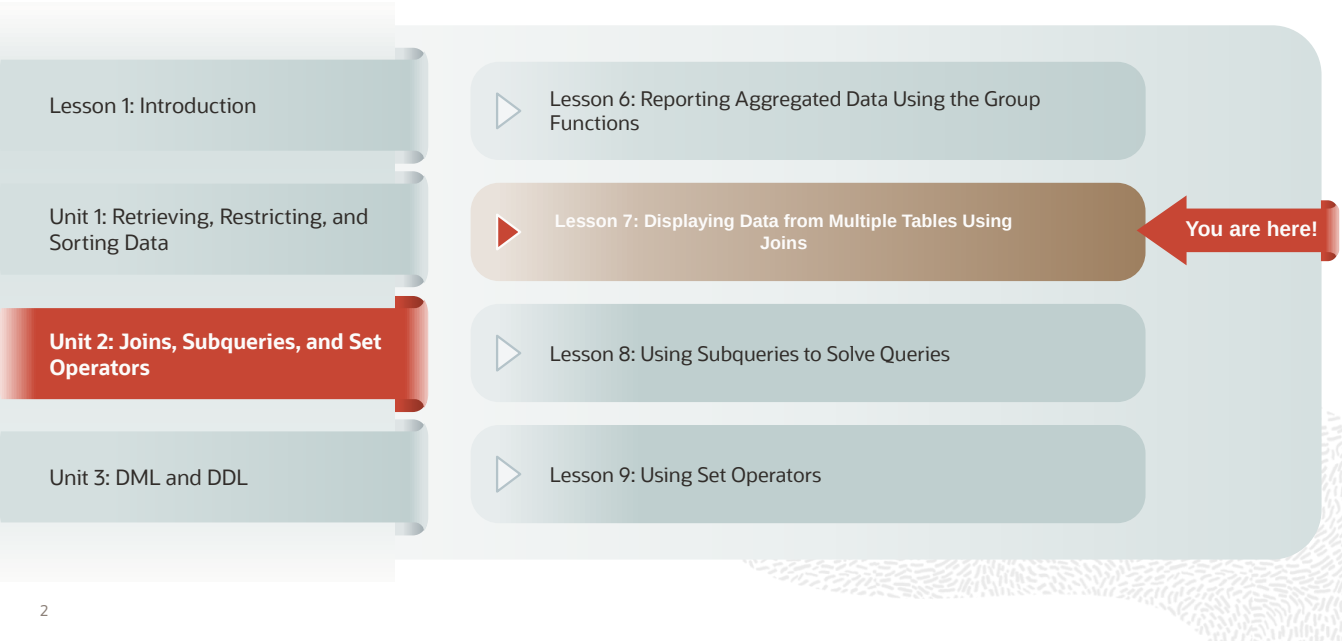
ORACLE

7

Displaying Data from Multiple Tables Using Joins

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 2, you will learn to use:

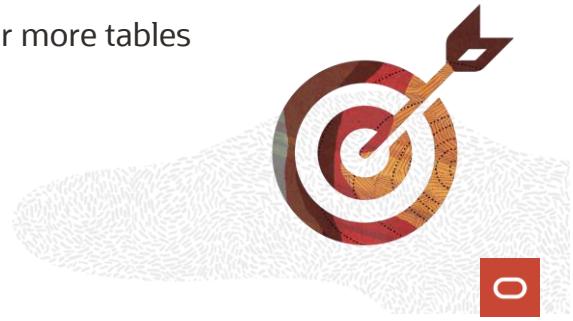
- SQL statements to query and display data from multiple tables using Joins
- Subqueries when the condition is unknown
- Group functions to aggregate data
- Set operators

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- 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



3

In this lesson, you learn how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

Note: For information about joins, refer to the “SQL Queries and Subqueries: Joins” section in *Oracle Database SQL Language Reference* for 18*c* database.

Oracle Internal & Oracle Academy Use Only

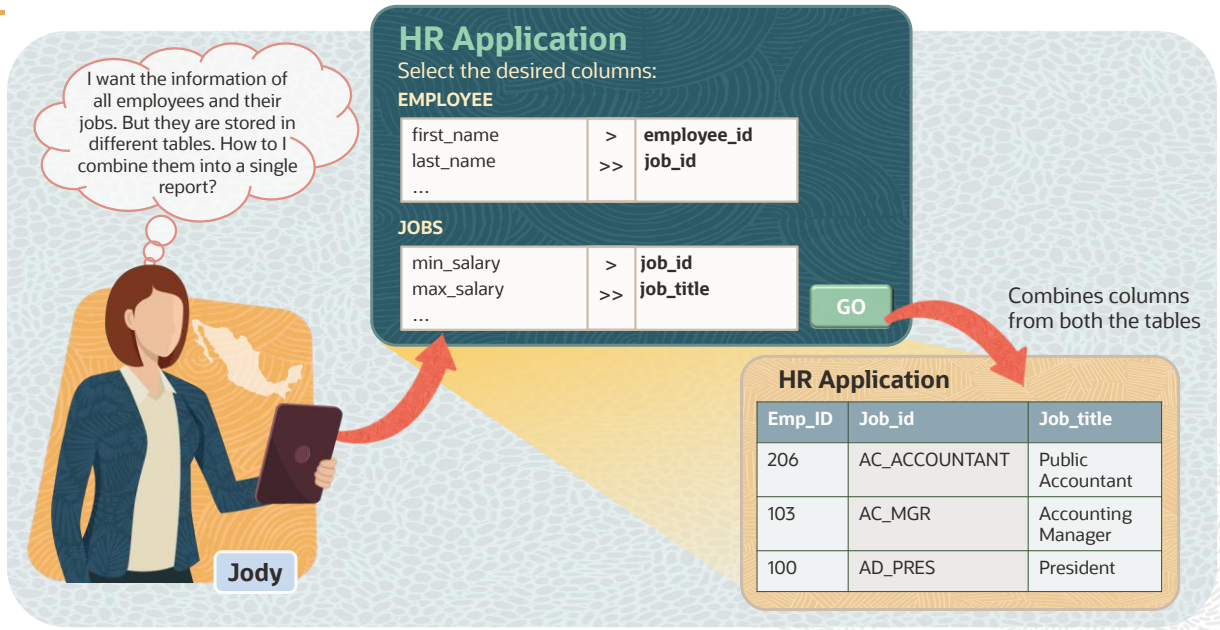
Lesson Agenda

- 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

4

Oracle Internal & Oracle Academy Use Only

Why Join?



5

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.

Oracle Internal & Oracle Academy Use Only

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	100	Steven	King	AD_PRES
2	101	Neena	Kochhar	AD_VP
3	102	Lex	De Haan	AD_VP
4	103	Alexander	Hunold	AC_MGR
5	104	Bruce	Ernst	IT_PROG
6	107	Diana	Lorentz	IT_PROG
7	124	Kevin	Mourgos	ST_MAN
8	141	Trenna	Rajs	ST_CLERK
9	142	Curtis	Davies	ST_CLERK
10	143	Randall	Matos	ST_CLERK

...

JOBS

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative

...

	EMPLOYEE_ID	JOB_ID	JOB_TITLE
1	206	AC_ACCOUNT	Public Accountant
2	205	AC_MGR	Accounting Manager
3	200	AD_ASST	Administration Assistant
4	100	AD_PRES	President
5	101	AD_VP	Administration Vice President
6	102	AD_VP	Administration Vice President
7	109	FI_ACCOUNT	Accountant

...

6

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.

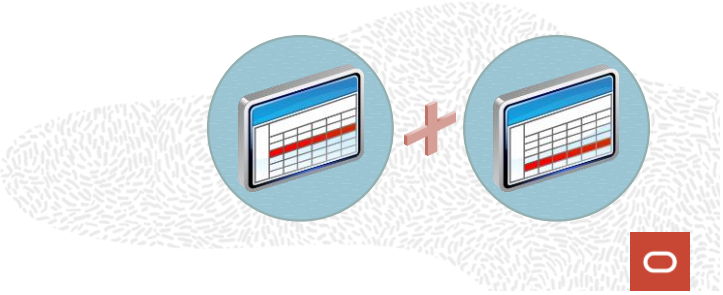
To produce the report, you need to link the `EMPLOYEES` and `JOBS` tables and access data from both of them.

Oracle Internal & Oracle Academy Use Only

Types of Joins

Joins that are compliant with the `ANSI` standard:

- 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



7

A join is a query that combines rows from two or more tables or views. Oracle Database performs a join whenever multiple tables appear in the `FROM` clause of the query.

Oracle Internal & Oracle Academy Use Only

Joining Tables Using SQL 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];
```

8

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 18*c* database.

Oracle Internal & Oracle Academy Use Only

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

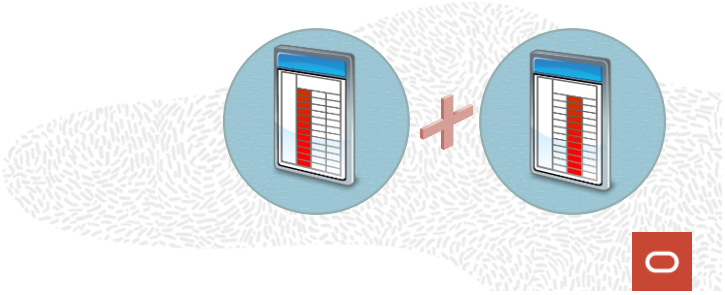
9

Oracle Internal & Oracle Academy Use Only

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;
```



10

You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the `NATURAL JOIN` clause.

Note: The join can happen on only those columns that have the same names and data types in both tables. If the columns have the same name but different data types, the `NATURAL JOIN` clause causes an error.

Oracle Internal & Oracle Academy Use Only

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	Kimberely	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

#	employee_ic	first_name	job_id	job_title
1	206	William	AC_ACC…	Public Accountant
2	205	Shelley	AC_MGR	Accounting Manager
3	200	Jennifer	AD_ASST	Administration Assistant
4	100	Steven	AD_PRES	President
5	101	Neena	AD_VP	Administration Vice President
6	102	Lex	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	176	Jonathon	SA_REP	Sales Representative
15	178	Kimberely	SA_REP	Sales Representative
16	141	Trenna	ST_CLERK	Stock Clerk
17	142	Curtis	ST_CLERK	Stock Clerk
18	143	Randall	ST_CLERK	Stock Clerk
19	144	Peter	ST_CLERK	Stock Clerk
20	124	Kevin	ST_MAN	Stock Manager

11

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);
```

Oracle Internal & Oracle Academy Use Only

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.

12

Natural joins use all columns with matching names and data types to join the tables. You can use the USING clause to specify only those columns that should be used for an equijoin.

An equijoin is a join containing an equality operator. A natural join is a type of equijoin.

Oracle Internal & Oracle Academy Use Only

Joining Column Names

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

...

DEPARTMENTS

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

Foreign key

Primary key

13

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.

Oracle Internal & Oracle Academy Use Only

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

#	employee_ic	last_name	location_ic	department_ic
1	103	Hunold	1400	60
2	104	Ernst	1400	60
3	107	Lorentz	1400	60
4	124	Mourgos	1500	50
5	141	Rajs	1500	50
6	142	Davies	1500	50
7	143	Matos	1500	50
8	144	Vargas	1500	50
...				
18	174	Abel	2500	80
19	176	Taylor	2500	80

14

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

Note that the EMPLOYEE_ID and LAST_NAME columns belong to the EMPLOYEES table, whereas LOCATION_ID belongs to the DEPARTMENTS table and the DEPARTMENT_ID column belongs to both these tables.

Oracle Internal & Oracle Academy Use Only

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.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

15

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.

Oracle Internal & Oracle Academy Use Only

Using Table Aliases with the USING Clause in Oracle

- 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
```

16

When joining with the `USING` clause in the Oracle Database, 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;
```

Oracle Internal & Oracle Academy Use Only

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.

17

Use the ON clause to specify a join condition. With this, you can specify join conditions separate from any search or filter conditions in the WHERE clause.

Oracle Internal & Oracle Academy Use Only

Retrieving Records with the ON Clause

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	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

...

#	employee_id	last_name	department_id	department_id	location_id
1	103	Hunold	60	60	1400
2	104	Ernst	60	60	1400
3	107	Lorentz	60	60	1400
4	124	Mourgos	50	50	1500
5	141	Rajs	50	50	1500
6	142	Davies	50	50	1500
7	143	Matos	50	50	1500
8	144	Vargas	50	50	1500
9	200	Whalen	10	10	1700
10	100	King	90	90	1700
11	101	Kochhar	90	90	1700

...

18

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`.

Oracle Internal & Oracle Academy Use Only

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

...

#	employee_ic	city	department_nam
1	149	Oxford	Sales
2	174	Oxford	Sales
3	176	Oxford	Sales
4	200	Seattle	Administration
5	100	Seattle	Executive
6	101	Seattle	Executive
7	102	Seattle	Executive
8	205	Seattle	Accounting
9	206	Seattle	Accounting

...

19

A three-way join is a join of three tables. Here, the first join to be performed is `employees JOIN departments`. The first join condition can reference columns in `employees` and `departments` but cannot reference columns in `locations`. The second join condition can reference columns from all three tables.

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

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

Oracle Internal & Oracle Academy Use Only

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 ;
```

20

You can apply additional conditions to the join.

The example shown performs a join on the EMPLOYEES and DEPARTMENTS tables and, in addition, displays only employees who have a manager ID of 149. To add additional conditions to the ON clause, you can add AND clauses. Alternatively, you can use a WHERE clause to apply additional conditions.

Both the queries produce the same output.

Oracle Internal & Oracle Academy Use Only

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

21

Oracle Internal & Oracle Academy Use Only

Joining a Table to Itself

EMPLOYEES (WORKER)

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

...

EMPLOYEES (MANAGER)

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

...

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

22

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.

Oracle Internal & Oracle Academy Use Only

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

...

#	emp	mgr
1	Kochhar	King
2	De Haan	King
3	Hunold	De Haan
4	Ernst	Hunold
5	Lorentz	Hunold
6	Mourgos	King
7	Rajs	Mourgos
8	Davies	Mourgos

23

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.

Oracle Internal & Oracle Academy Use Only

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

24

Oracle Internal & Oracle Academy Use Only

Nonequijoins

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.

25

A nonequijoin is a join condition containing something other than an equality operator.

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

Oracle Internal & Oracle Academy Use Only

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

#	last_name	salary	grade_level
1	King	24000.00	E
2	Kochhar	17000.00	E
3	De Haan	17000.00	E
4	Hunold	9000.00	C
5	Ernst	6000.00	C
6	Lorentz	4200.00	B
7	Mourgos	5800.00	B
8	Rajs	3500.00	B
9	Davies	3100.00	B
10	Matos	2600.00	A

26

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.

Oracle Internal & Oracle Academy Use Only

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

27

Oracle Internal & Oracle Academy Use Only

Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

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

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.

28

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:

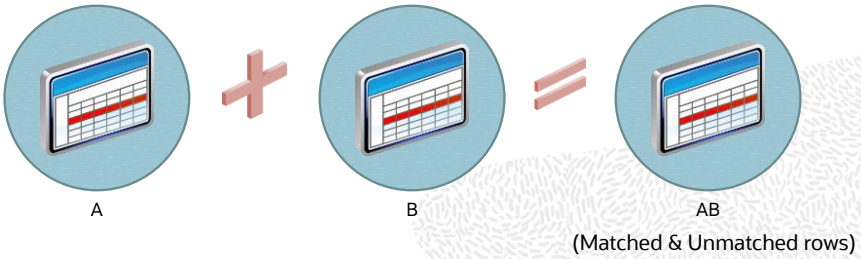
- Department ID 190, because there are no employees with that department ID recorded in the EMPLOYEES table
- The employee with the last name of Grant, because this employee has not been assigned a department ID

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.

Oracle Internal & Oracle Academy Use Only

INNER Versus OUTER Joins

- 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.
- In Oracle, 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.



29

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

Note: MySQL does not support FULL OUTER JOIN.

Oracle Internal & Oracle Academy Use Only

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)

#	last_name	department_i	department_nam
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT
5	Ernst	60	IT
...			
14	Taylor	80	Sales
15	Grant	NULL	NULL
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting

30

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

Basically, a `LEFT OUTER JOIN` will return all the rows that matches between the two tables on the given condition and also returns the unmatched rows of the left table, that is, the table mentioned first in the SQL statement.

Oracle Internal & Oracle Academy Use Only

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

#	last_name	department_i	department_nam
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Mourgos	50	Shipping
5	Rajs	50	Shipping
6	Davies	50	Shipping
7	Matos	50	Shipping
8	Vargas	50	Shipping
...			
18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	NULL	190	Contracting

31

This query retrieves all the rows in the `departments` table, which is the table at the right, even if there is no match in the `employees` table.

Basically, a `RIGHT OUTER JOIN` will return all the rows that matches between the two tables on the given condition and also returns the unmatched rows of the right table, that is, the table mentioned second in the SQL statement.

Oracle Internal & Oracle Academy Use Only

FULL OUTER JOIN in Oracle

```
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

32

This query retrieves all rows in the `EMPLOYEES` table, even if there is no match in the `DEPARTMENTS` table. It also retrieves all rows in the `DEPARTMENTS` table, even if there is no match in the `EMPLOYEES` table.

Basically, a `FULL OUTER JOIN` will return all the rows that match between the two tables on the given condition and also returns the unmatched rows of both the tables.

This function is not supported in MySQL.

Oracle Internal & Oracle Academy Use Only

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

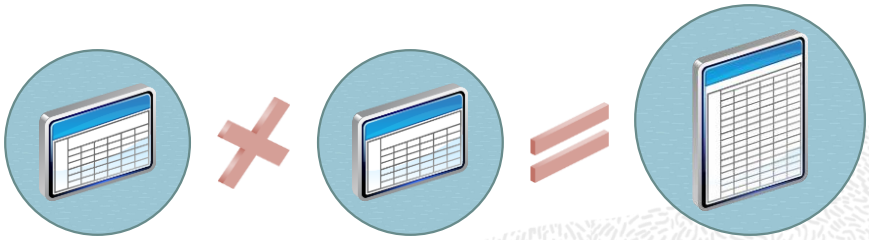
33

Oracle Internal & Oracle Academy Use Only

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



34

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.

Oracle Internal & Oracle Academy Use Only

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

35

A Cartesian product is generated if a join condition is omitted. The example in the slide displays the employee ID, department ID, and location ID from the `EMPLOYEES` and `DEPARTMENTS` tables. Because no join condition was specified, all rows (20 rows) from the `EMPLOYEES` table are joined with all rows (8 rows) in the `DEPARTMENTS` table, thereby generating 160 rows in the output.

Oracle Internal & Oracle Academy Use Only

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

#	last_name	department_nam
1	Abel	Administration
2	Abel	Marketing
3	Abel	Shipping
4	Abel	IT
5	Abel	Sales
...		
156	Zlotkey	IT
157	Zlotkey	Sales
158	Zlotkey	Executive
159	Zlotkey	Accounting
160	Zlotkey	Contracting

36

The example in the slide produces a Cartesian product of `EMPLOYEES` and `DEPARTMENTS` tables.

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

Oracle Internal & Oracle Academy Use Only

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

37

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.

Oracle Internal & Oracle Academy Use Only

Practice 7: Overview

This practice covers the following topics:

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

38

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

Oracle Internal & Oracle Academy Use Only

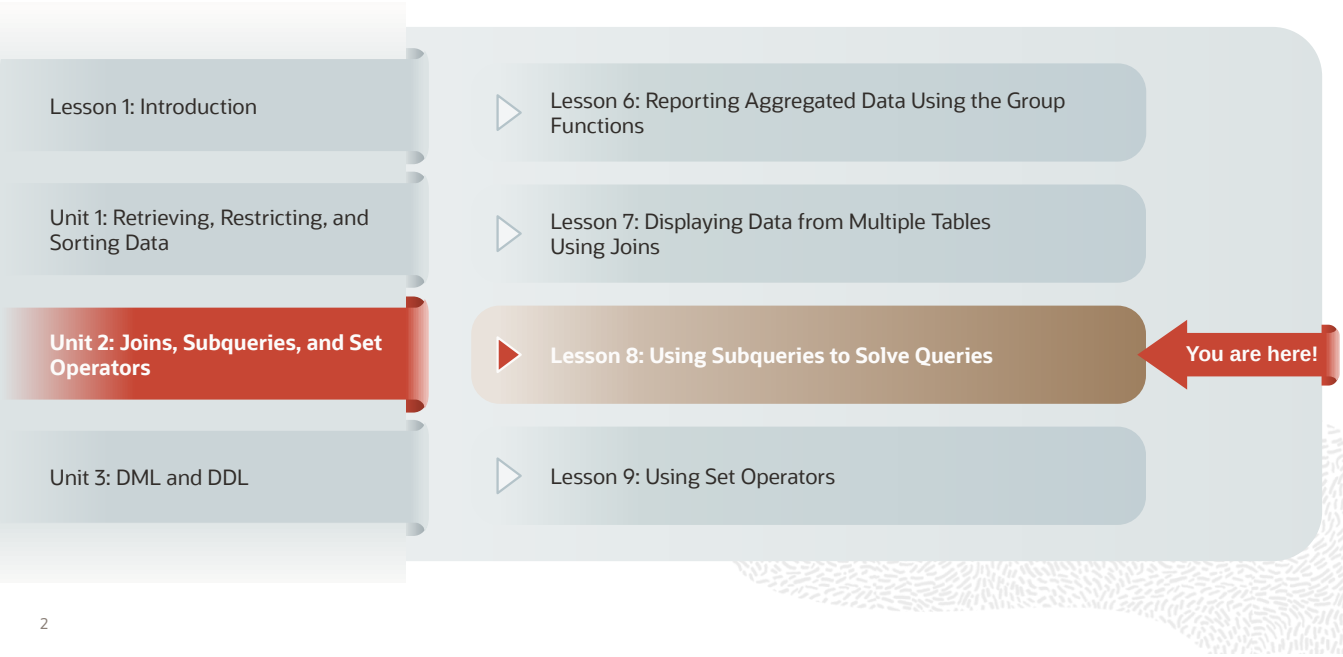
ORACLE

8

Using Subqueries to Solve Queries

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 2, you will learn to use:

- SQL statements to query and display data from multiple tables by using joins
- Subqueries when the condition is unknown
- Group functions to aggregate data
- Set operators

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Define subqueries
- Describe the types of problems that subqueries can solve
- Identify the types of subqueries
- Write single-row, multiple-row, multiple-column subqueries

3

In this lesson, you learn about the more advanced features of the `SELECT` statement. You can write subqueries in the `WHERE` clause of another SQL statement to obtain values based on an unknown conditional value. This lesson also covers single-row subqueries, multiple-row subqueries, and multiple-column subqueries.

Oracle Internal & Oracle Academy Use Only

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

4

Oracle Internal & Oracle Academy Use Only

Using a Subquery to Solve a Problem

5

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*).

Oracle Internal & Oracle Academy Use Only

Subquery Syntax

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

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

Main Query

Subquery

6

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

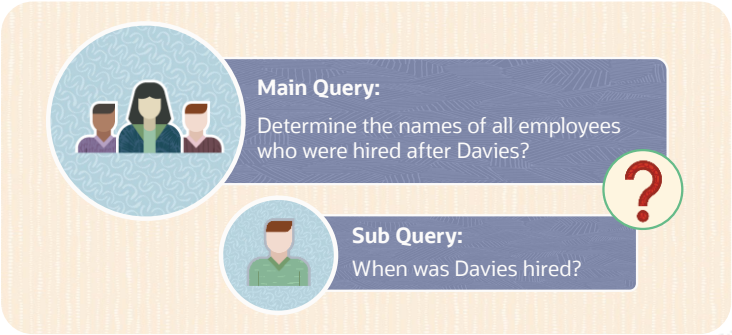
In the syntax:

`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.

Oracle Internal & Oracle Academy Use Only

Using a Subquery




```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date > (SELECT hire_date
                    FROM   employees
                    WHERE  last_name = 'Davies');
```

7

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.

Oracle Internal & Oracle Academy Use Only

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.

8

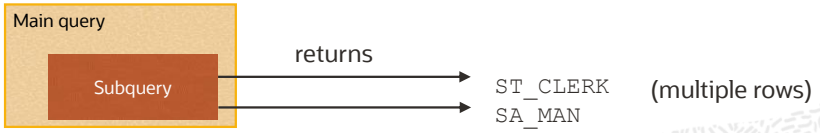
Two classes of comparison conditions are used in subqueries: single-row operators and multiple-row operators.

Oracle Internal & Oracle Academy Use Only

Types of Subqueries

- Single-row subquery

- Multiple-row subquery



9

- **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.

Oracle Internal & Oracle Academy Use Only

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

10

Oracle Internal & Oracle Academy Use Only

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

11

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

Oracle Internal & Oracle Academy Use Only

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =                    SA_REP
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary >                    8600
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Taylor');
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

#	last_name	job_id	salary
1	Abel	SA_REP	11000.00

12

A `SELECT` statement can be considered as a query block. The example in the slide displays employees who do the same job as “Taylor” but earn more salary than him.

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

Both inner queries return single values (`SA_REP` and `8600`, respectively), so this SQL statement is called a single-row subquery.

Note: The outer and inner queries can get data from different tables.

Oracle Internal & Oracle Academy Use Only

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

#	last_name	job_id	salary
1	Vargas	ST_CLERK	2500.00

13

You can display data from a main query by using a group function in a subquery to return a single row. The subquery is in parentheses and is placed after the comparison condition.

The example in the slide displays the employee last name, job `ID`, and salary of all employees whose salary is equal to the minimum salary. The `MIN` group function returns a single value (`2500`) to the outer query.

Oracle Internal & Oracle Academy Use Only

HAVING Clause with Subqueries

The database 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) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```

2500

	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

#	department_i	MIN(salary)
1	NULL	7000.00
2	10	4400.00
3	20	6000.00
4	60	4200.00
5	80	8600.00
6	90	17000.00
7	110	8300.00

14

You can use subqueries not only in the WHERE clause but also in the HAVING clause. The 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);
```

Oracle Internal & Oracle Academy Use Only

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

```
Error Code: 1242. Subquery returns more than 1 row
```

15

A common error with subqueries occurs when more than one row is returned for a single-row subquery.

In the SQL statement in the slide, the subquery contains a `GROUP BY` clause, which implies that the subquery will return multiple rows, one for each group that it finds. In this case, the results of the subquery are `4400`, `6000`, `2500`, `4200`, `7000`, `17000`, and `8300`.

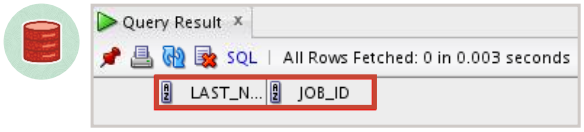
The outer query takes those results and uses them in its `WHERE` clause. The `WHERE` clause contains an equal (=) operator, a single-row comparison operator that expects only one value. The = operator cannot accept more than one value from the subquery and, therefore, generates the error.

To correct this error, change the = operator to `IN`.

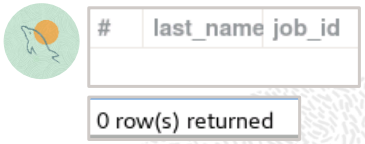
Oracle Internal & Oracle Academy Use Only

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`."



16

Another common problem occurs with subqueries when no rows are returned by the inner query.

In the SQL statement in the slide, the subquery contains a `WHERE` clause. Presumably, the intention is to find the employee who works as an 'Architect'. The statement is correct, but selects no rows when executed because there is no job titled 'Architect'. Therefore, the subquery returns no rows.

The outer query takes the results of the subquery (null) and uses these results in its `WHERE` clause. The outer query finds no employee with a job `ID` equal to `NULL` and so returns no rows. Even if a job existed with a value of null, the row is not returned because comparison of two null values yields a null; therefore, the `WHERE` condition is not true.

Oracle Internal & Oracle Academy Use Only

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

17

Oracle Internal & Oracle Academy Use Only

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.

18

Subqueries that return more than one row are called multiple-row subqueries. You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values:

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (SELECT   MIN(salary)
                  FROM     employees
                  GROUP BY department_id);
```

Example

Find the employees who earn the same salary as the minimum salary of any department.

The inner query is executed first, producing a query result. The main query block is then processed and uses the values that were returned by the inner query to complete its search condition. In fact, the main query appears to the database server as follows:

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (2500, 4200, 4400, 6000, 7000, 8300,
                  8600, 17000);
```

Oracle Internal & Oracle Academy Use Only

Using the ANY Operator in Multiple-Row Subqueries

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

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
	143	Matos	ST_CLERK	2600
	142	Davies	ST_CLERK	3100
	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

#	employee_ic	last_name	job_id	salary
1	124	Mourgos	ST_MAN	5800.00
2	141	[illegible]	ST_CLERK	3500.00
3	142	[illegible]es	ST_CLERK	3100.00
4	143	[illegible]tos	ST_CLERK	2600.00
5	144	Vargas	ST_CLERK	2500.00
6	176	Taylor	SA_REP	8600.00
7	178	Grant	SA_REP	7000.00
8	200	Whalen	AD_ASST	4400.00
9	202	Fay	MK_REP	6000.00
10	206	Gietz	AC_ACC...	8300.00
*	NULL	NULL	NULL	NULL

19

The ANY operator (and its synonym, the SOME operator) compares a value to *each* value returned by a subquery. The example in the slide displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is $9,000.

- <ANY means less than the maximum.
- >ANY means more than the minimum.
- =ANY is equivalent to IN.

Oracle Internal & Oracle Academy Use Only

Using the ALL Operator in Multiple-Row Subqueries

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

9000, 6000, 4200

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

#	employee_ic	last_name	job_id	salary
1	141	Rajs	ST_CLERK	3500.00
2	142	Davies	ST_CLERK	3100.00
3	143	Matos	ST_CLERK	2600.00
4	144	Vargas	ST_CLERK	2500.00
*	NULL	NULL	NULL	NULL

20

The ALL operator compares a value to *every* value returned by a subquery. The example in the slide displays employees who are not IT programmers and whose salary is less than that of all IT programmers.

- >ALL means more than the maximum.
- <ALL means less than the minimum.
- The NOT operator can be used with IN, ANY, and ALL operators.

Oracle Internal & Oracle Academy Use Only

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);
```

21

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);
```

Oracle Internal & Oracle Academy Use Only

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

#	first_name	department_id	salary
1	Jennifer	10	4400.00
2	Pat	20	6000.00
3	Peter	50	2500.00
4	Diana	60	4200.00
5	Jonathon	80	8600.00
6	Neena	90	17000.00
7	Lex	90	17000.00
8	William	110	8300.00

22

The example in the slide shows a multiple-column subquery in which the subquery returns more than one column.

The inner query is executed first, and it returns the lowest `salary` and `department_id` for each department. The main query block is then processed and uses the values that were returned by the inner query to complete its search condition.

Note: The employee with the first name Kimberely is not returned in the result because the `department_id` for the employee is `NULL`.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

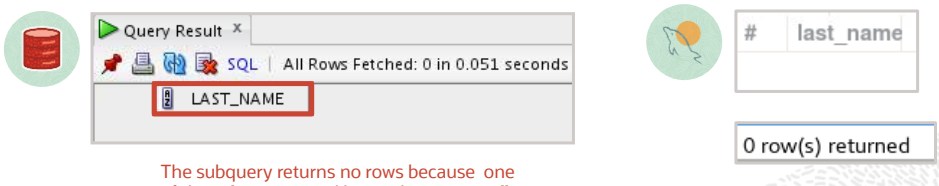
- Subquery: Types, syntax, and guidelines
- Single-row subqueries:
  - Group functions in a subquery
  - `HAVING` clause with subqueries
- Multiple-row subqueries
  - Using `ALL` or `ANY` operator
- Multiple-column subqueries
- Null values in a subquery

23

Oracle Internal & Oracle Academy Use Only

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.

24

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);
```

Oracle Internal & Oracle Academy Use Only

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

25

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

Oracle Internal & Oracle Academy Use Only

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

26

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.

Oracle Internal & Oracle Academy Use Only

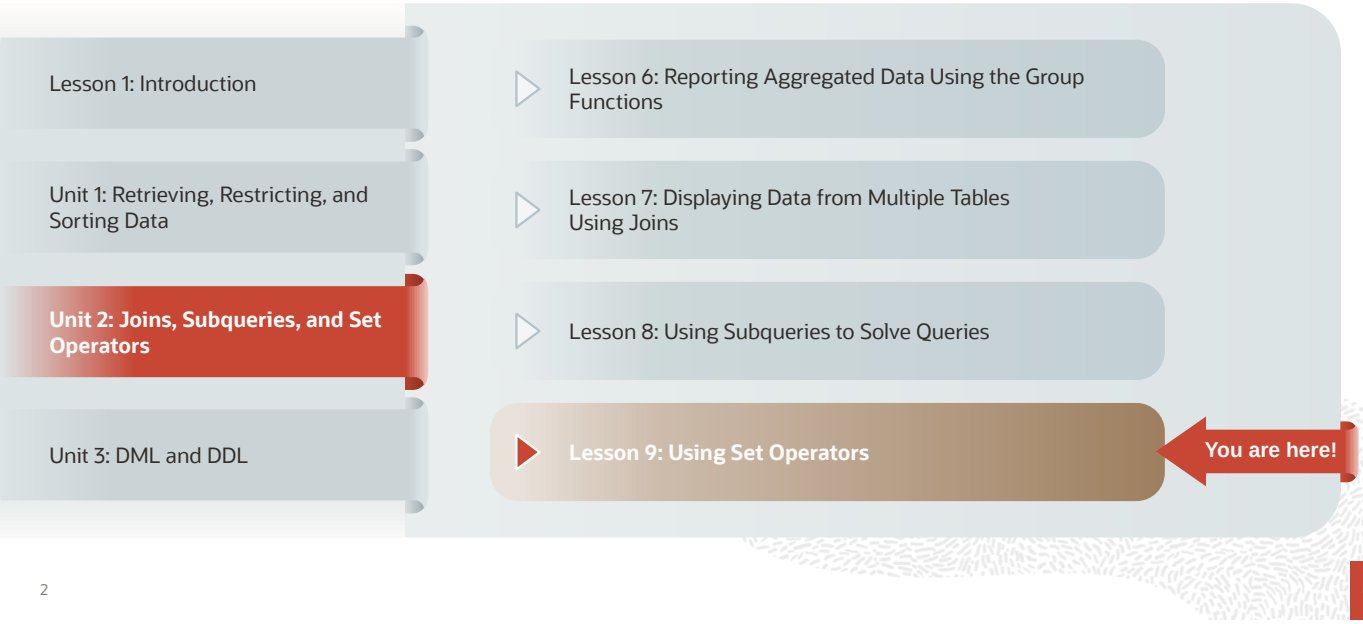
ORACLE

9

Using Set Operators

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 2, you will learn to use:

- SQL statements to query and display data from multiple tables using Joins
- Subqueries when the condition is unknown
- Group functions to aggregate data
- Set operators

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

3

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

Oracle Internal & Oracle Academy Use Only

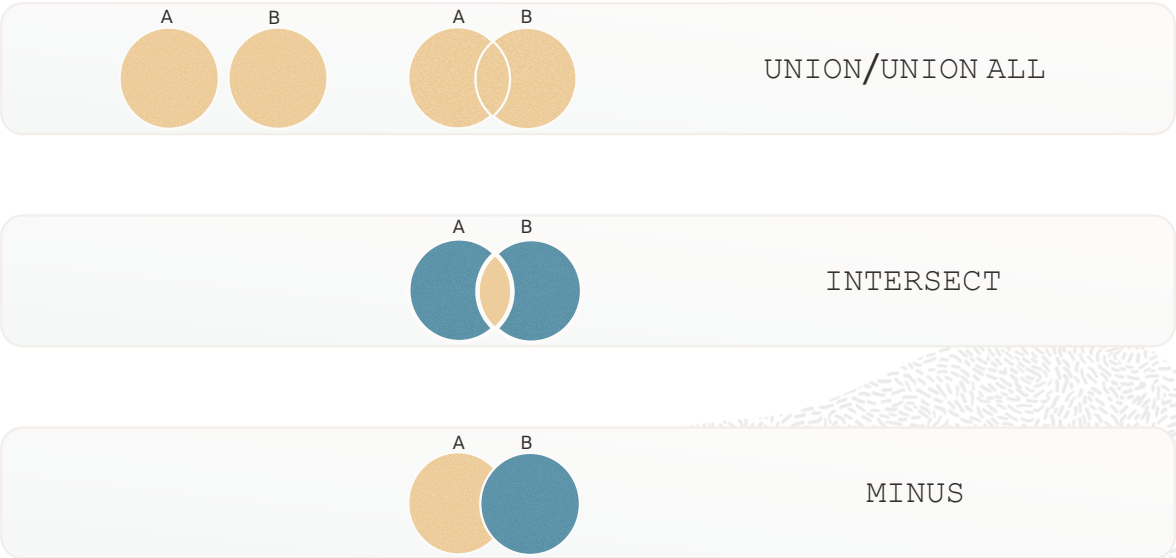
Lesson Agenda

- 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

4

Oracle Internal & Oracle Academy Use Only

Set Operators



5

Set operators combine the results of two or more component queries into one result. Queries containing set operators are called *compound queries*.

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

All set operators have equal precedence. If a SQL statement contains multiple set operators, the Oracle server evaluates them from left (top) to right (bottom), if no parentheses explicitly specify another order. You should use parentheses to specify the order of evaluation explicitly in queries that use the `INTERSECT` operator with other set operators.

Oracle Internal & Oracle Academy Use Only

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.

6

- 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 Internal & Oracle Academy Use Only

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`.

7

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.

Oracle Internal & Oracle Academy Use Only

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

8

Oracle Internal & Oracle Academy Use Only

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

9

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.

Oracle Internal & Oracle Academy Use Only

```
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)
```

Oracle Internal & Oracle Academy Use Only

```
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)
```

Oracle Internal & Oracle Academy Use Only

```
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

Oracle Internal & Oracle Academy Use Only

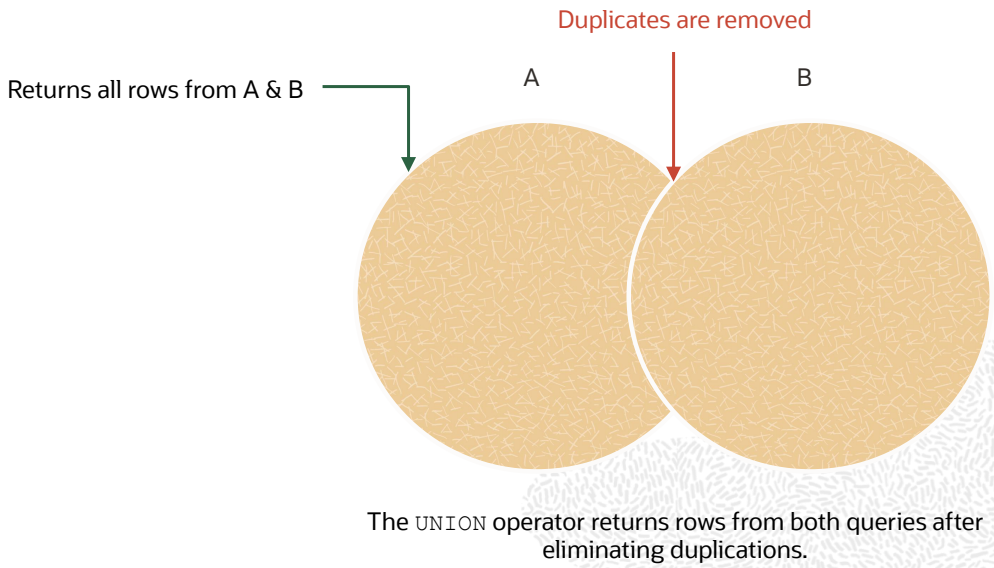
Lesson Agenda

- 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

13

Oracle Internal & Oracle Academy Use Only

UNION Operator



14

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.

Oracle Internal & Oracle Academy Use Only

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

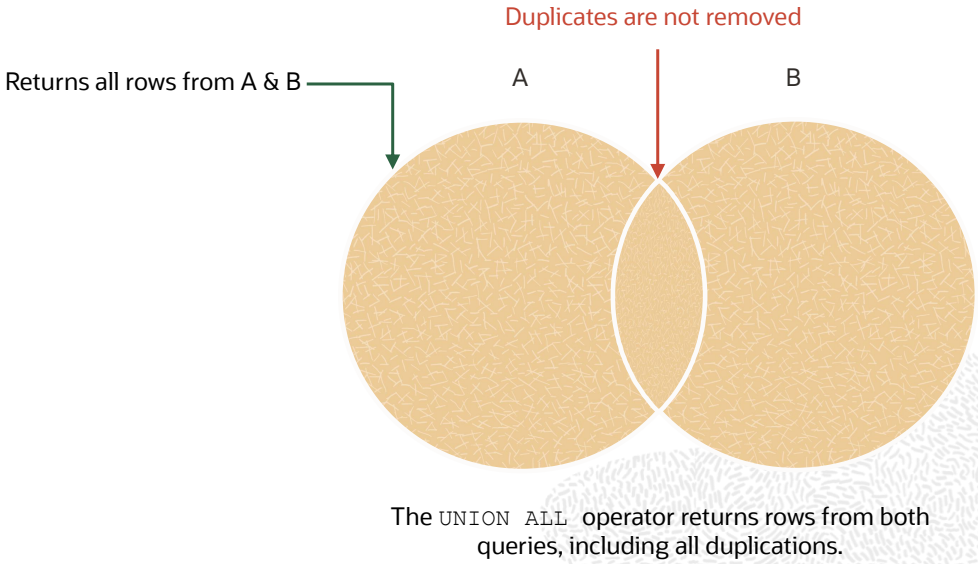
#	job_id
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	IT_PROG
7	MK_MAN
8	MK_REP
9	SA_MAN
10	SA_REP
11	ST_CLERK
12	ST_MAN
13	FI_MGR
14	FI_ACCOUNT
15	PU_MAN
16	PU_CLERK

15

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.

Oracle Internal & Oracle Academy Use Only

UNION ALL Operator



16

Use the UNION ALL operator to return all rows from multiple queries.

Guidelines

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL: Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.

Oracle Internal & Oracle Academy Use Only

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

#	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	90
8	AD_VP	80
9	AD_VP	90

28	SA_REP	NULL
29	SA_REP	80
30	SA_REP	80
31	ST_CLERK	50
32	ST_CLERK	50
33	ST_CLERK	50
34	ST_CLERK	50
35	ST_MAN	50

17

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.

Oracle Internal & Oracle Academy Use Only

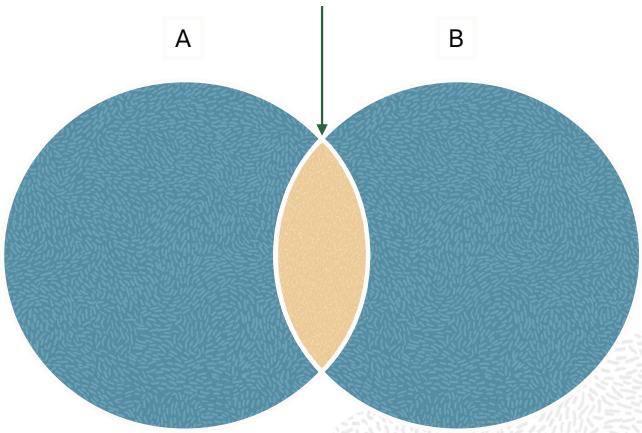
Lesson Agenda

- 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

18

Oracle Internal & Oracle Academy Use Only

INTERSECT Operator



The INTERSECT operator returns rows that are common to both queries.

19

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.

Oracle Internal & Oracle Academy Use Only

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

20

In the example in this slide, the query returns only those records that have the same values in the selected columns in both tables.

Oracle Internal & Oracle Academy Use Only

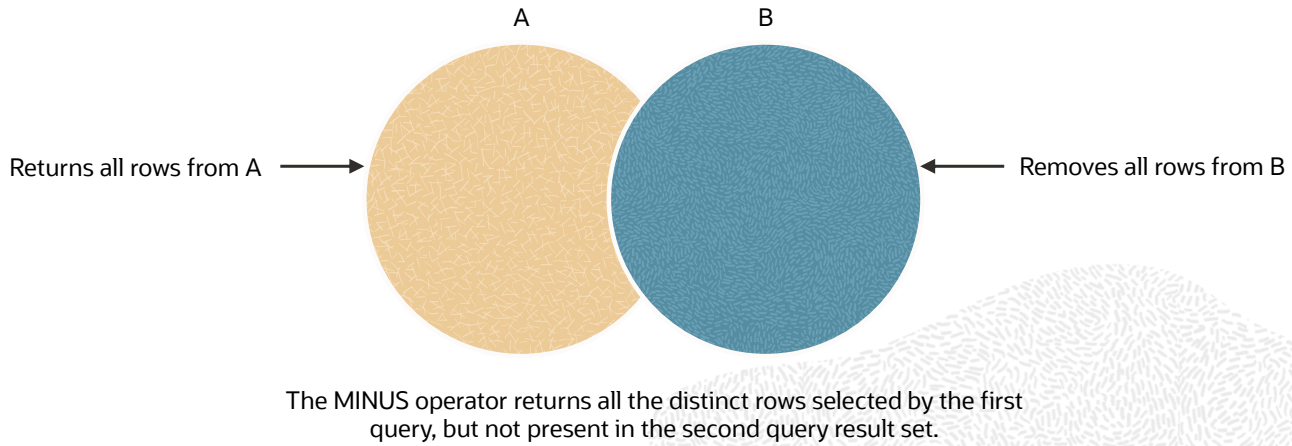
Lesson Agenda

- 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

21

Oracle Internal & Oracle Academy Use Only

MINUS Operator



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

22

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.

Oracle Internal & Oracle Academy Use Only

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

23

In the example in the slide, the manager IDs and job IDs in the RETIRED_EMLOYEES table are subtracted from those in the EMPLOYEES table. The results set displays the employees remaining after the subtraction; they are represented by rows that exist in the EMPLOYEES table, but do not exist in the RETIRED_EMPLOYEES table. These are the records of the employees who work in the sales department and are being managed by managers who have not managed any of the retired employees.

Oracle Internal & Oracle Academy Use Only

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

24

Oracle Internal & Oracle Academy Use Only

Matching SELECT Statements in Oracle

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;
```

25

Because the expressions in the SELECT lists of the queries must match in number, you can use the dummy columns and the data type conversion functions to comply with this rule.

To match the column list explicitly, you can insert NULL columns at the missing positions so as to match the count and data type of selected columns in each SELECT statement.

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

Oracle Internal & Oracle Academy Use Only

Matching the SELECT Statement: Example in Oracle

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	Alexandera	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	Del	PU_MAN	(null)

...

26

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.

Oracle Internal & Oracle Academy Use Only

Matching SELECT Statements in MySQL

You must match the data type (using the CAST function) when columns do not exist in one or the other table.

```
SELECT location_id, department_name 'Department',
   CAST(NULL AS CHAR) 'Warehouse location'
FROM departments
UNION
SELECT location_id, CAST(NULL AS CHAR),
   state_province
FROM locations;
```

#	location_i	Department	Warehouse locatio
1	1700	Administration	NULL
2	1800	Marketing	NULL
3	1500	Shipping	NULL
4	1400	IT	NULL
5	2500	Sales	NULL
6	1700	Executive	NULL
7	1700	Accounting	NULL
8	1700	Contracting	NULL
9	1500	NULL	California
10	1800	NULL	Ontario
11	2500	NULL	Oxford
12	1400	NULL	Texas
13	1700	NULL	Washington

27

Because the expressions in the SELECT lists of the queries must match in number, you can use the dummy columns and the data type conversion functions to comply with this rule.

To match the column list explicitly, you can insert NULL columns at the missing positions so as to match the count and data type of selected columns in each SELECT statement.

In the slide, the name Warehouse location is given as the dummy column heading. The CAST function is used in the first query to match the VARCHAR data type of the state_province column that is retrieved by the second query. Similarly, the CAST function in the second query is used to match the VARCHAR data type of the department_name column that is retrieved by the first query.

Oracle Internal & Oracle Academy Use Only

Matching the SELECT Statement: Example in MySQL

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, CAST(NULL AS DATE)
FROM retired_employees;
```

#	first_name	job_id	Hire Date
1	Steven	AD_PRES	2011-06-17
2	Neena	AD_VP	2009-09-21
3	Lex	AD_VP	2009-01-13
4	Alexander	IT_PROG	2014-01-03
5	Bruce	IT_PROG	2015-05-21
6	Diana	IT_PROG	2015-02-07
7	Kevin	ST_MAN	2015-11-16
8	Trenna	ST_CLERK	2011-10-17
9	Curtis	ST_CLERK	2013-01-29

...

28	Rahul	FI_MGR	NULL
29	Dany	FI_ACCOUNT	NULL
30	James	FI_ACCOUNT	NULL
31	Shana	FI_ACCOUNT	NULL
32	Peter	FI_ACCOUNT	NULL
33	Lui	FI_ACCOUNT	NULL
34	Del	PU_MAN	NULL
35	Alex	PU_CLERK	NULL

28

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.

Oracle Internal & Oracle Academy Use Only

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

29

Oracle Internal & Oracle Academy Use Only

Using the ORDER BY Clause in Set Operations in Oracle

- 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.

30

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.

Oracle Internal & Oracle Academy Use Only

Using the ORDER BY Clause in Set Operations in Oracle: Example

Display the employee ID and job ID of all current and retired employees, sorted by job ID.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   retired_employees
ORDER BY 2;
```

	EMPLOYEE_ID	JOB_ID
1	206	AC_ACCOUNT
2	205	AC_MGR
3	200	AD_ASST
4	100	AD_PRES
5	301	AD_PRES
6	101	AD_VP
7	102	AD_VP
8	302	AD_VP
9	303	AD_VP
10	309	FI_ACCOUNT
11	310	FI_ACCOUNT
12	311	FI_ACCOUNT
13	312	FI_ACCOUNT
14	313	FI_ACCOUNT
...		
28	174	SA_REP
29	176	SA_REP
30	178	SA_REP
31	141	ST_CLERK
32	142	ST_CLERK
33	143	ST_CLERK
34	144	ST_CLERK
35	124	ST_MAN

31

In the example shown on the slide, the output is sorted in the ascending order of `job_id`.

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.

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.

Oracle Internal & Oracle Academy Use Only

Using the ORDER BY Clause with UNION in MySQL

- To use an ORDER BY clause to sort the entire UNION result, place the ORDER BY clause only once at the end of the compound query.
- The ORDER BY clause uses the columns of the first SELECT query.
- If a column to be sorted is aliased, the ORDER BY clause must use the alias rather than the column name.

You can use the ORDER BY clause to sort the entire result set in a compound query. Place the ORDER BY clause at the end of the query. In the ORDER BY clause, you can refer to a column alias or column name from the first SELECT statement. If the sort column is aliased, you must use the alias rather than the column name. You can use the column position in the ORDER BY clause, but this use is deprecated.

Note: You can use an ORDER BY clause in each SELECT statement, but it has no effect on the sort order of the result set, but rather is used by the LIMIT clause. To apply an ORDER BY clause to an individual SELECT statement, enclose the statement in parentheses, with the ORDER BY and LIMIT clauses inside the parentheses. For example, the following query selects and combines the first 10 employees from each table and sorts the results by Job ID.

```
(SELECT employee_id, job_id
FROM   employees ORDER BY employee_id LIMIT 10)
UNION
(SELECT employee_id, job_id
FROM   retired_employees ORDER BY employee_id LIMIT 10)
ORDER BY job_id;
```

Oracle Internal & Oracle Academy Use Only

Using the ORDER BY Clause with UNION: Example in MySQL

Display the employee ID and job ID of all current and retired employees, sorted by job ID.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   retired_employees
ORDER BY job_id;
```

#	employee_ic	job_id
1	206	AC_ACCOUNT
2	205	AC_MGR
3	200	AD_ASST
4	100	AD_PRES
5	301	AD_PRES
6	101	AD_VP
7	102	AD_VP
8	303	AD_VP
9	302	AD_VP

...

28	178	SA_REP
29	176	SA_REP
30	174	SA_REP
31	144	ST_CLERK
32	143	ST_CLERK
33	142	ST_CLERK
34	141	ST_CLERK
35	124	ST_MAN

33

In the example shown in the slide, the output is sorted in the ascending order of `job_id`.

Oracle Internal & Oracle Academy Use Only

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

34

- 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.

Oracle Internal & Oracle Academy Use Only

Practice 9: Overview

In this practice, you create reports by using:

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

35

In this practice, you write queries using set operators.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

10A

Managing Tables Using DML Statements in Oracle

Oracle Internal & Oracle Academy Use Only

Course Roadmap

In Unit 3, you learn how to manage data in tables using data manipulation language (DML) statements. You also learn how to create and manage database objects using data definition language (DDL) statements.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

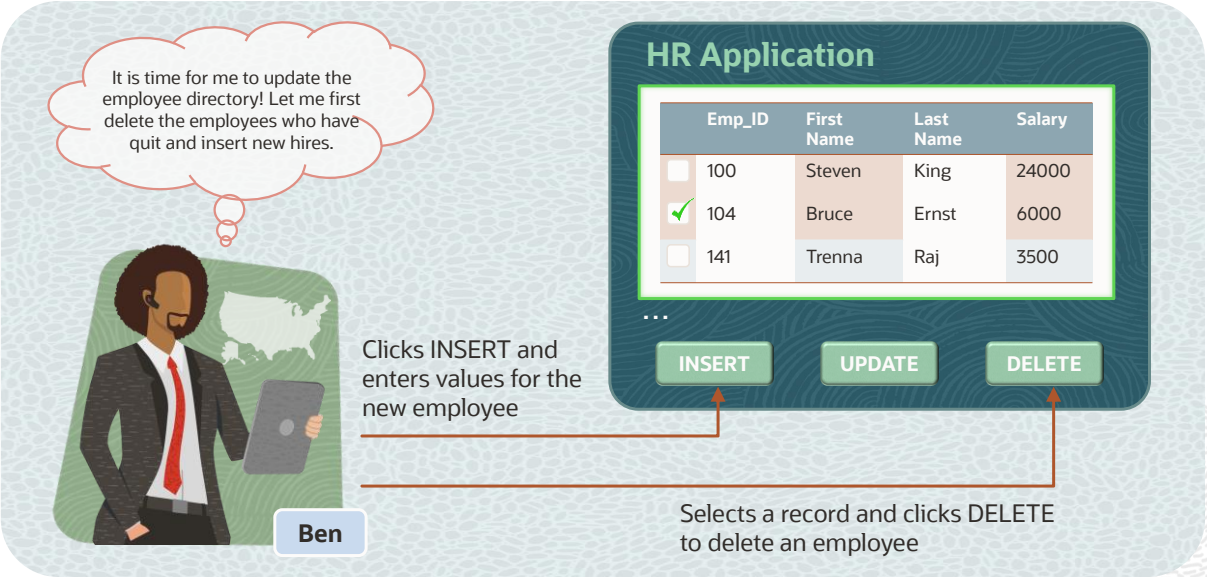
- Describe each data manipulation language (DML) statement
- Control transactions

3

In this lesson, you learn how to use data manipulation language (DML) statements to insert rows into a table, update existing rows in a table, and delete existing rows from a table. You also learn how to control transactions with the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

Oracle Internal & Oracle Academy Use Only

HR Application Scenario



4

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 `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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

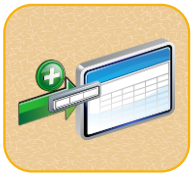
- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - FOR `UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

5

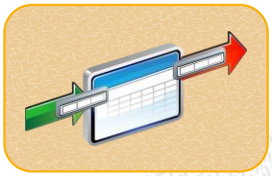
Oracle Internal & Oracle Academy Use Only

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.



Insert



Update

Delete

6

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 database 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.

Oracle Internal & Oracle Academy Use Only

Adding a New Row to a Table

DEPARTMENTS

70	Public Relations	100	1700

New row

	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 a new row into the DEPARTMENTS table.

	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

7

The graphic in the slide illustrates the addition of a new department record to the DEPARTMENTS table.

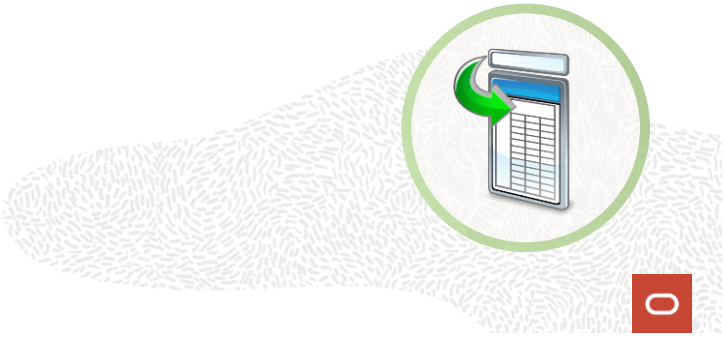
Oracle Internal & Oracle Academy Use Only

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.



8

You can add new rows to a table by issuing the INSERT statement.

In the syntax:

table	Is the name of the table
column	Is the name of the column in the table to populate
value	Is the corresponding value for the column

Note: This statement with the VALUES clause adds only one row at a time to a table.

Oracle Internal & Oracle Academy Use Only

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.

9

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.

Oracle Internal & Oracle Academy Use Only

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.
```

10

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.

Oracle Internal & Oracle Academy Use Only

Inserting Special Values

The CURRENT_DATE function records the current date and time in Oracle.

```
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',
                       CURRENT_DATE, 'AC_ACCOUNT', 6900,
                       NULL, 205, 110);
```

```
1 row inserted.
```

11

You can use functions to enter special values in your table.

The example in the slide records information for employee Popp in the EMPLOYEES table. It supplies the current date and time in the HIRE_DATE column. It uses the CURRENT_DATE function that returns the current date in the session time zone. You can also use the USER function when inserting rows in a table. The USER function records the current username.

Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

Oracle Internal & Oracle Academy Use Only

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, 70);
```

```
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	70

12

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.

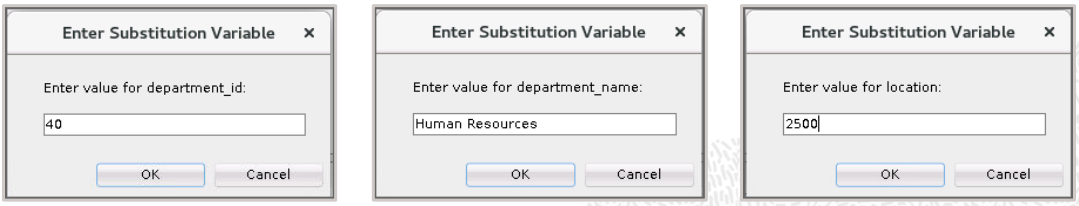
The example in the slide records information for employee Raphealy in the EMPLOYEES table. It sets the HIRE_DATE column to be February 3, 2016.

Oracle Internal & Oracle Academy Use Only

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);
```



13

You can save commands with substitution variables to a file and execute the commands in the file. The example in the slide records information for a department in the DEPARTMENTS table.

Run the script file and you are prompted for input for each of the ampersand (&) substitution variables. After entering a value for the substitution variable, click the OK button. The values that you input are then substituted into the statement. This enables you to run the same script file over and over, but supply a different set of values each time you run it.

Oracle Internal & Oracle Academy Use Only

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.
- Insert all the rows returned by the subquery in the table, sales_reps.

14

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:

table	Is the name of the table
column	Is the name of the column in the table to populate
subquery	Is the subquery that returns rows to the table

The number of columns and their data types in the column list of the INSERT clause must match the number of values and their data types in the subquery. 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;
```

Oracle Internal & Oracle Academy Use Only

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

15

Oracle Internal & Oracle Academy Use Only

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)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

16

The slide illustrates changing the department number for employees in department 60 to department 80.

Oracle Internal & Oracle Academy Use Only

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).

17

You can modify the existing values in a table by using the `UPDATE` statement.

In the syntax:

table	Is the name of the table
column	Is the name of the column in the table to populate
value	Is the corresponding value or subquery for the column
Condition	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 19c 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 having the same last name.

Oracle Internal & Oracle Academy Use Only

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.

18

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.

Oracle Internal & Oracle Academy Use Only

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.
```

19

You can update multiple columns in the `SET` clause of an `UPDATE` statement by writing multiple subqueries. The syntax is as follows:

```
UPDATE table
SET     column  =
                                (SELECT column
                                 FROM table
                                 WHERE condition)
       [ ,
        column  =
                                (SELECT column
                                 FROM table
                                 WHERE condition)]
[WHERE  condition ]          ;
```

Oracle Internal & Oracle Academy Use Only

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.
```

20

You can use the subqueries in the UPDATE statements to update values in a table. The example in the slide updates the COPY_EMP table based on the values from the EMPLOYEES table. It changes the department number to employee 100's current department number for all the employees whose job ID is the same as employee 200's job ID.

Oracle Internal & Oracle Academy Use Only

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

21

Oracle Internal & Oracle Academy Use Only

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

22

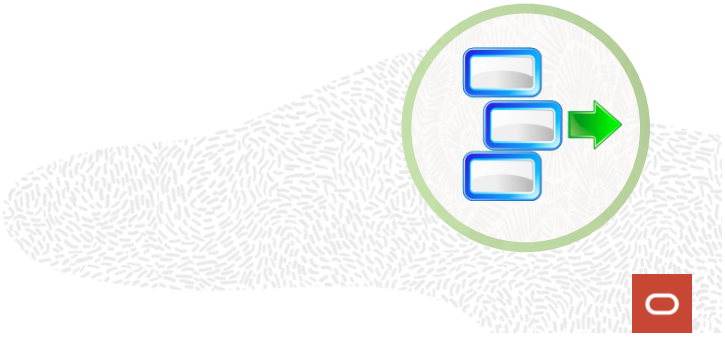
The contracting department has been removed from the DEPARTMENTS table (assuming no constraints on the DEPARTMENTS table are violated), as shown in the graphic in the slide.

Oracle Internal & Oracle Academy Use Only

DELETE Statement

You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM]     table
[WHERE       condition];
```



23

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 19c database.

Oracle Internal & Oracle Academy Use Only

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
```

24

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 = 113;

DELETE FROM  departments WHERE department_id IN (30, 40);
```

Oracle Internal & Oracle Academy Use Only

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.
```

25

You can use the subqueries to delete rows from a table based on values from another table. The example in the slide deletes all the employees in a department, where the department name contains the string `Public`.

The subquery searches the `DEPARTMENTS` table to find the department number based on the department name containing the string `Public`. The subquery then feeds the department number to the main query, which deletes rows of data from the `EMPLOYEES` table.

Oracle Internal & Oracle Academy Use Only

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
-

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

26

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.”

Oracle Internal & Oracle Academy Use Only

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

27

Oracle Internal & Oracle Academy Use Only

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

28

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.

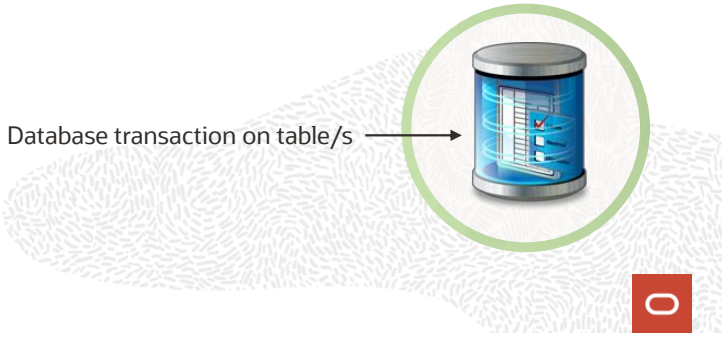
Transaction Types

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

Oracle Internal & Oracle Academy Use Only

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.



29

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.

Oracle Internal & Oracle Academy Use Only

Advantages of the COMMIT and ROLLBACK Statements

Using COMMIT and ROLLBACK statements, you can:

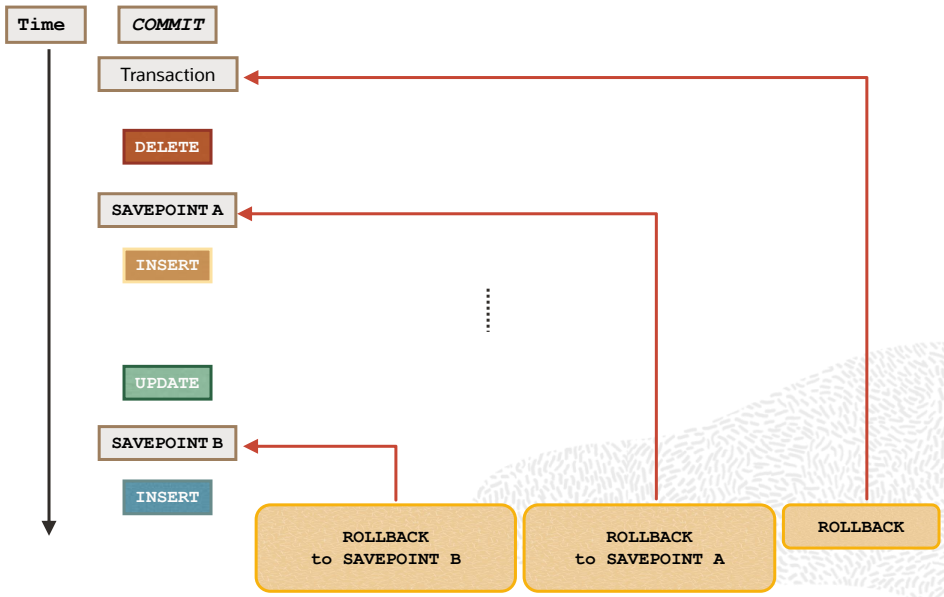
- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

30

With the COMMIT and ROLLBACK statements, you have control over making changes to the data permanent.

Oracle Internal & Oracle Academy Use Only

Explicit Transaction Control Statements



31

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 *name*	SAVEPOINT *name* marks a savepoint within the current transaction.
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes.
ROLLBACK TO *SAVEPOINT name*	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.

Oracle Internal & Oracle Academy Use Only

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

32

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

Note that if you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

Oracle Internal & Oracle Academy Use Only

Implicit Transaction Processing

- An automatic commit occurs when:
  - A DDL statement is issued
  - A DCL statement is issued
  - There is 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.

33

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

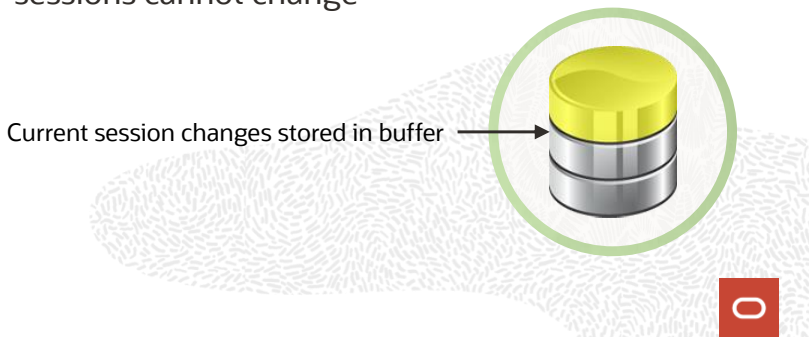
When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to the state at the time of the last commit. In this way, the Oracle server protects the integrity of the 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.

Oracle Internal & Oracle Academy Use Only

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.



34

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.

Oracle Internal & Oracle Academy Use Only

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.

35

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.

Oracle Internal & Oracle Academy Use Only

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.
```

36

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;
```

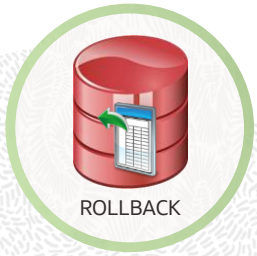
Oracle Internal & Oracle Academy Use Only

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 ;
```



37

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.

Oracle Internal & Oracle Academy Use Only

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.
```

38

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.

Note: Refer to code_ex_10.sql in the labs/sql1_oracle/code_ex folder to create the test table.

Oracle Internal & Oracle Academy Use Only

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.

39

If a single DML statement fails during the execution of a transaction, its effect is undone by a statement-level implicit rollback; however, the changes made by the previous DML statements in the transaction are not discarded. These can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

Oracle Internal & Oracle Academy Use Only

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

40

Oracle Internal & Oracle Academy Use Only

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

41

Database users access the database in two ways:

- Read operations (`SELECT` statement)
- Write operations (`INSERT`, `UPDATE`, `DELETE` statements)

You need read consistency so that:

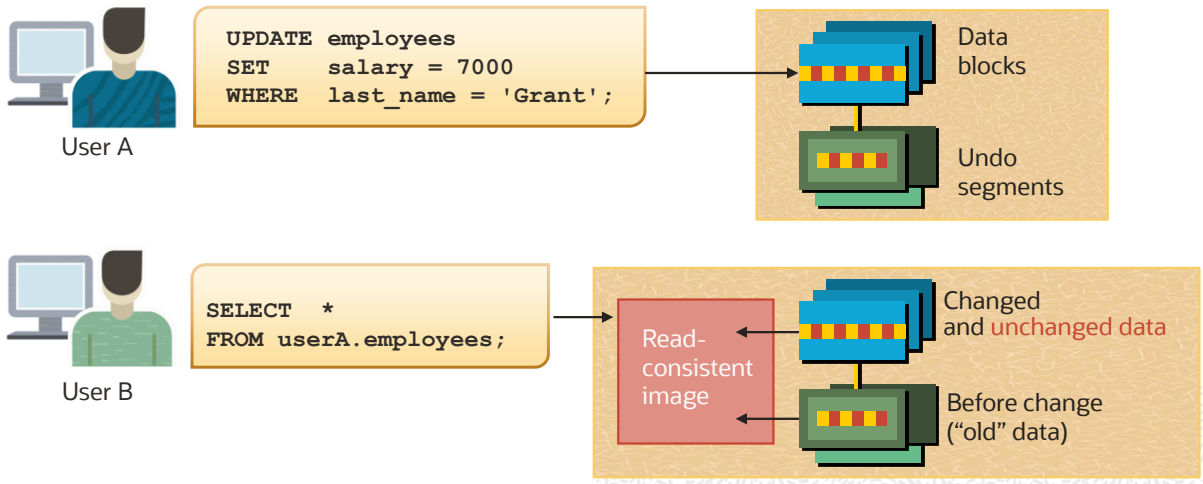
- The database reader and writer are ensured a consistent view of data
- Readers do not view data that is in the process of being changed
- Writers are ensured that the changes to the database are done in a consistent manner
- Changes made by one writer do not disrupt or conflict with the changes being 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.

Oracle Internal & Oracle Academy Use Only

Implementing Read Consistency



42

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 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.

Oracle Internal & Oracle Academy Use Only

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

43

Oracle Internal & Oracle Academy Use Only

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.

44

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.

Oracle Internal & Oracle Academy Use Only

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.

45

In the example in the slide, the statement locks rows in the EMPLOYEES table with JOB_ID set to ST_CLERK and LOCATION_ID set to 1500 and locks rows in the DEPARTMENTS table with departments in LOCATION_ID set as 1500.

You can use the FOR UPDATE OF *column_name* to qualify the column that you intend to change. The OF list of the FOR UPDATE clause does not restrict you to changing only those columns of the selected rows. Locks are still placed on all rows; if you simply state FOR UPDATE in the query and do not include one or more columns after the OF keyword, the database will lock all identified rows across all the tables listed in the FROM clause.

The following statement locks only those rows in the EMPLOYEES table with ST_CLERK located in LOCATION_ID 1500. No rows are locked in the DEPARTMENTS table:

```
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 OF e.salary
ORDER BY e.employee_id;
```

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;
```

Oracle Internal & Oracle Academy Use Only

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];
```

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 19c 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.

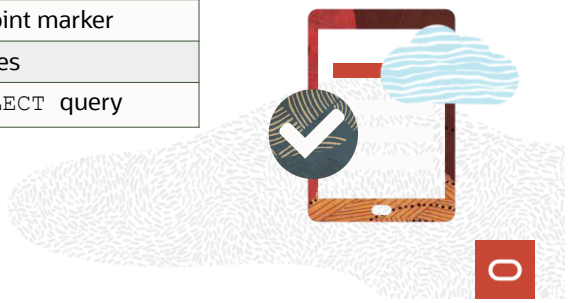
```
LOCK TABLE employees
IN EXCLUSIVE MODE
NOWAIT;
```

Oracle Internal & Oracle Academy Use Only

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



47

In this lesson, you should have learned how to manipulate data in the Oracle database by using the `INSERT`, `UPDATE`, `DELETE`, and `TRUNCATE` statements, as well as how to control data changes by using the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements. You should have also learned how to use the `FOR UPDATE` clause of the `SELECT` statement to lock rows for your changes only.

Remember that the Oracle server guarantees a consistent view of data at all times.

Oracle Internal & Oracle Academy Use Only

Practice 10a: Overview

This practice covers the following topics:

- Inserting rows into the tables
- Updating and deleting rows in the table
- Controlling transactions

48

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.

Oracle Internal & Oracle Academy Use Only

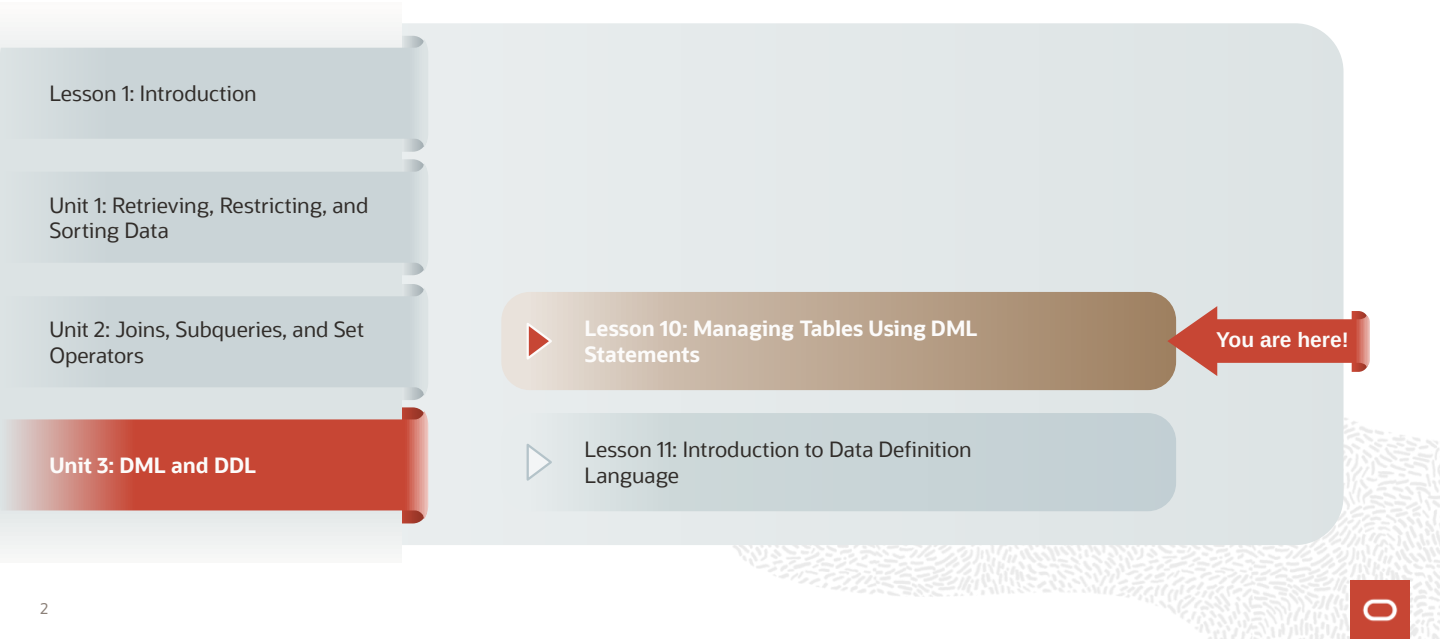
ORACLE

10B

Managing Tables Using DML Statements in MySQL

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 3, you learn how to manage data in tables using data manipulation language (DML) statements. You also learn how to create and manage database objects using data definition language (DDL) statements.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

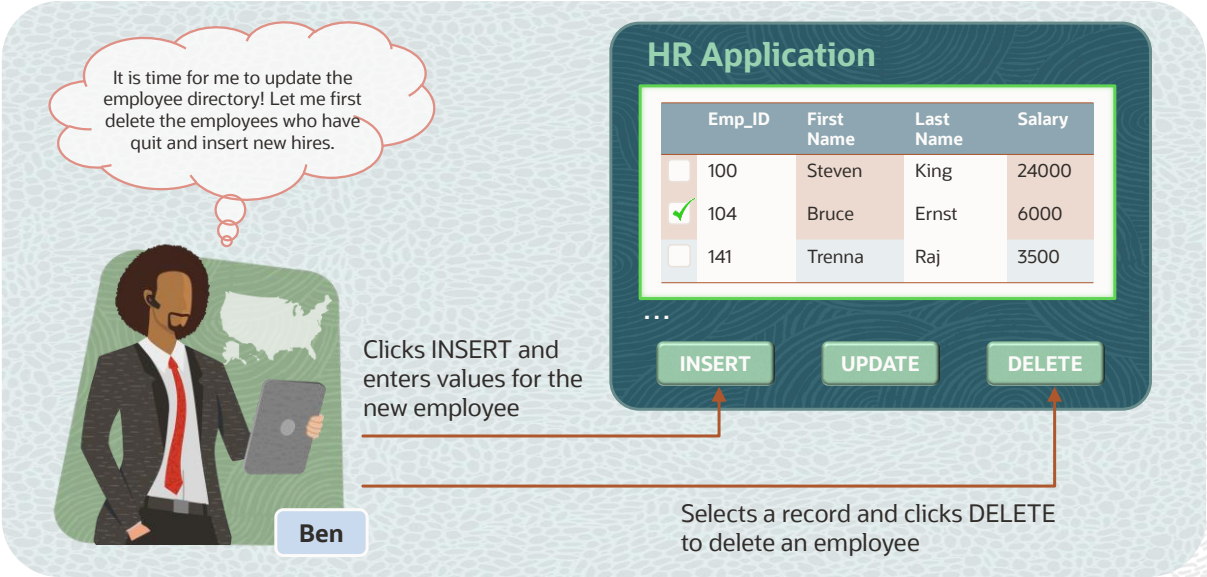
- Describe each data manipulation language (DML) statement
- Control transactions

3

In this lesson, you learn how to use data manipulation language (DML) statements to insert rows into a table, update existing rows in a table, and delete existing rows from a table. You also learn how to control transactions with the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

Oracle Internal & Oracle Academy Use Only

HR Application Scenario



4

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 `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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

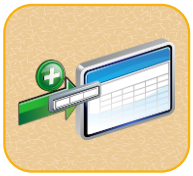
- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transaction control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

5

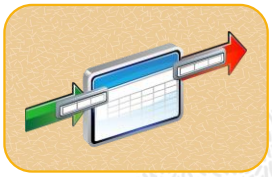
Oracle Internal & Oracle Academy Use Only

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.



Insert



Update

Delete

6

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 database 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.

Oracle Internal & Oracle Academy Use Only

Adding a New Row to a Table

`departments`

	70 Public Relations	100	1700

New row

	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 a new row into the `departments` table.

	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

7

The graphic in the slide illustrates the addition of a new department record to the `departments` table.

Oracle Internal & Oracle Academy Use Only

INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement:

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]),
              (value [, value...]),
              ...
              (value [, value...]);
```

- Each comma-separated list of values, enclosed in parentheses, inserts a new row.
- Enclose character and date data in quotation marks.

8

You can add new rows to a table by issuing the INSERT statement.

In the syntax:

- ***table***: The name of the table. The table name is required.
- ***column***: The names of the columns in the table to populate. The column names are optional. As described below, the values listed depend on whether and how column names are provided.
- ***value***: The corresponding value for each column. The list of values for each row are enclosed within parentheses and separated by commas. If the column names are provided, the list of values must be in the same order as the list of corresponding column names. If the column names are not provided, each list of values must include all columns in the order provided by the DESCRIBE statement.

Oracle Internal & Oracle Academy Use Only

Inserting New Rows: Listing Column Names

If you include the list of column names after the table name:

- You can list the columns in any order
- The values must be listed in the same order as the columns they populate
- The list of values has to include the same number of values as the number of columns

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700),
       (150, 'Shareholder Services', 100, 1700);
```

```
2 row(s) affected
Records: 2  Duplicates: 0  Warnings: 0
```

9

The list of column names in parentheses after the table name in the `INSERT` statement is optional. If you include the list of column names, it does not have to be in the same order as the default order of the columns in the table. The list of column names can be in any order, but the values listed must be in the same order as the list of columns and has to have the same number of values as the list of columns.

The example in this slide inserts two rows. You could also create two separate `INSERT` statements, one for each row.

Oracle Internal & Oracle Academy Use Only

Inserting New Rows: Omitting Column Names

If you omit the list of column names after the table name:

- You must include a value for every column in the table
- The values must be listed in the same order as the default order of the columns, as displayed by a `DESCRIBE table_name;` statement

```
INSERT INTO departments
VALUES (120, 'Treasury', 100, 1700),
       (130, 'Corporate Tax', 100, 1700),
       (140, 'Control and Credit', 100, 1700);
```

```
3 row(s) affected
Records: 3  Duplicates: 0  Warnings: 0
```

10

The default order of columns is the order that columns are displayed as output from a `SELECT * FROM table_name;` query. You can find the default order by issuing a `DESCRIBE table_name;` statement.

Oracle Internal & Oracle Academy Use Only

Inserting Rows with Null Values

- **Implicit method:** If you omit a column from the column list, the value is set to the default value for the column or `NULL` if the column does not have a default value.

```
INSERT INTO   departments (department_id,
                           department_name)
VALUES        (30, 'Purchasing');
```

1 row(s) affected

- **Explicit method:** You can specify the `NULL` keyword in the `VALUES` list.

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
```

1 row(s) affected

11

If you include a column list, but omit one or more of the columns, the value is set to the default value for the column or NULL. You learn about setting default values in the lesson titled "Introduction to Data Definition Language in MySQL." Be sure that you can use null values in the targeted column by verifying the `NULL` status with the `DESCRIBE` command.

Oracle Internal & Oracle Academy Use Only

Inserting Special Values in MySQL

The `CURDATE()` function records the current date and time in MySQL.

```
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',
                       CURDATE(), 'AC_ACCOUNT', 6900,
                       NULL, 205, 110);
```

1 row(s) affected

12

You can use functions to enter special values in your table.

The example in the slide records information for employee Popp in the `employees` table. It supplies the current date and time in the `hire_date` column. It uses the `CURDATE()` function that returns the current date.

Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

Oracle Internal & Oracle Academy Use Only

Inserting Specific Date and Time Values in MySQL

- Add a new employee.

```
INSERT INTO employees
VALUES      (115,
             'Alexander', 'Khoo',
             'AKHOO', '515.127.4562',
             '2011-05-18',
             'SA_REP', 3100, 0.2, 100, 60);
```

1 row(s) affected

- Verify your addition.

```
SELECT * FROM   employees WHERE  employee_id = 115;
```

#	employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	commission_pct	manager_id	department_id
1	115	Alexander	Khoo	AKHOO	515.127.4562	2011-05-18	SA_REP	3100.00	0.20	100	60
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

13

You can enter dates using the default date format, just as you do in `WHERE` conditions or other references to dates.

Oracle Internal & Oracle Academy Use Only

Inserting and Reformatting Specific Date and Time Values in MySQL

- Add a new employee.

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             STR_TO_DATE('FEB 3, 2016', '%M %d, %Y'),
             'SA_REP', 11000, 0.2, 100, 70);
```

1 row(s) affected

- Verify your addition.

```
SELECT * FROM   employees WHERE  employee_id = 114;
```

#	employee_ic	first_name	last_name	email	phone_numbe	hire_date	job_id	salary	commission_p	manager_id	department_id
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	2016-02-03	SA_REP	11000.00	0.20	100	70
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

14

In MySQL, if you want to enter the date in a format other than the default format, you must use the STR_`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, 2016, but stores and displays it in the default date format.

Oracle Internal & Oracle Academy Use Only

Copying Rows from Another Table

- Write your INSERT statement with a subquery:

```
INSERT INTO copy_emp
  SELECT * FROM   employees;
```

23 row(s) affected
Records: 23 Duplicates: 0 Warnings: 0

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees WHERE  job_id LIKE '%REP%';
```

6 row(s) affected
Records: 6 Duplicates: 0 Warnings: 0

- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.
- It inserts all the rows returned by the subquery into the table.

15

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:

table	Is the name of the table
column	Is the name of the column in the table to populate
subquery	Is the subquery that returns rows to the table

The number of columns and their data types in the column list of the INSERT clause must match the number of values and their data types in the subquery. Zero or more rows are added depending on the number of rows returned by the subquery.

Oracle Internal & Oracle Academy Use Only

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

16

Oracle Internal & Oracle Academy Use Only

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)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

17

The slide illustrates changing the department number for employees in department 60 to department 80.

Oracle Internal & Oracle Academy Use Only

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).

18

You can modify the existing values in a table by using the UPDATE statement.

In the syntax:

table	Is the name of the table
column	Is the name of the column in the table to populate
value	Is the corresponding value or subquery for the column
Condition	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 19c 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.

Oracle Internal & Oracle Academy Use Only

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(s) affected
Rows matched: 1  Changed: 1  Warnings: 0
```

- Values for all the rows in the table are modified if you omit the `WHERE` clause:

```
UPDATE     copy_emp
SET        department_id = 110;
```

```
20 row(s) affected
Rows matched: 23  Changed: 20  Warnings: 0
```

- Specify `SET` *`column_name`*= NULL to update a column value to `NULL`.

19

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.

Oracle Internal & Oracle Academy Use Only

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(s) affected
Rows matched: 1  Changed: 1  Warnings: 0
```

20

You can use the subqueries in the `UPDATE` statements to update values in a table. The example in the slide updates the `copy_emp` table based on the values from the `employees` table. It changes the department number to employee 100's current department number (90) for all the employees whose job ID is the same as employee 200's job `ID`. In MySQL, you cannot update a table and select from the same table in a subquery.

Oracle Internal & Oracle Academy Use Only

Quiz

In an `UPDATE` statement, you can use a subquery for which of the following clauses?

a. The `SET` clause only

b. The `WHERE` clause only

c. Either the `SET` clause or the `WHERE` clause or both clauses

d. Neither the `SET` clause nor the `WHERE` clause

21

Answer: c

Oracle Internal & Oracle Academy Use Only

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

22

Oracle Internal & Oracle Academy Use Only

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

23

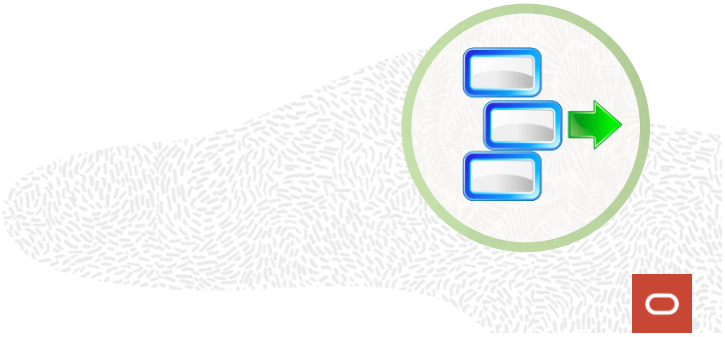
The contracting department has been removed from the `departments` table (assuming no constraints on the `departments` table are violated), as shown in the graphic in the slide.

Oracle Internal & Oracle Academy Use Only

DELETE Statement

You can remove existing rows from a table by using the `DELETE` statement:

```
DELETE [FROM]     table
[WHERE       condition];
```



24

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 19c database.

Oracle Internal & Oracle Academy Use Only

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
DELETE FROM departments
WHERE  department_name = 'Finance';
```

1 row(s) affected

- All rows in the table are deleted if you omit the WHERE clause:

```
DELETE FROM  copy_emp;
```

23 row(s) affected

25

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 = 115;

DELETE FROM  departments WHERE department_id IN (130, 140);
```

Oracle Internal & Oracle Academy Use Only

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(s) affected

26

You can use the subqueries to delete rows from a table based on values from another table. The example in the slide deletes all the employees in a department, where the department name contains the string `Public`.

The subquery searches the `departments` table to find the department number based on the department name containing the string `Public`. The subquery then feeds the department number to the main query, which deletes rows of data from the `employees` table.

Oracle Internal & Oracle Academy Use Only

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;
```

27

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.”

Oracle Internal & Oracle Academy Use Only

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

28

Oracle Internal & Oracle Academy Use Only

Multiple-Statement Transactions

- A collection of statements that are treated as a single unit:
  - Groups multiple statements
  - Is useful when multiple clients can access data from the same table at the same time
  - Is important when statements affect the same or related data
- All or none of the steps succeed:
  - Execute if all steps are successful.
  - Cancel if any steps cause errors or do not complete.

29

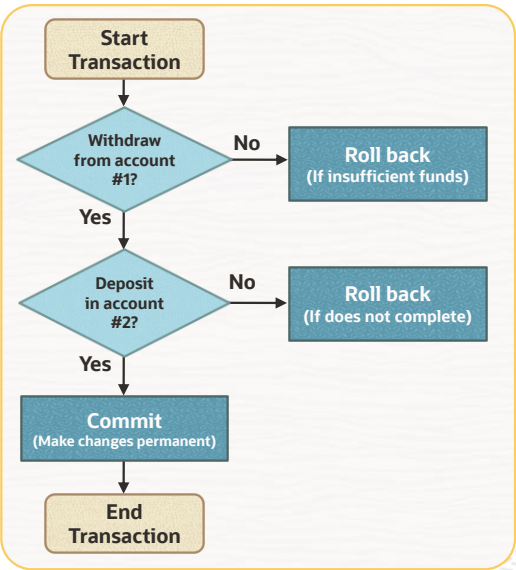
A transaction lets you execute one or more SQL statements as a single unit of work, so that either all or none of the statements succeed. This happens independently of work being done by any other transactions. If all the statements succeed, you commit the transaction to record the effect permanently in the database. If an error occurs during the transaction, you roll it back to cancel it. Any statements executed up to the point of canceling within the transaction are undone, leaving the database in the state it was in before the transaction.

Note: MySQL supports transactions only for tables that use a transactional storage engine (such as InnoDB). These statements have no noticeable effect on tables managed by non-transactional storage engines.

Oracle Internal & Oracle Academy Use Only

Transaction Diagram

Example of a banking transaction:



The diagram in the slide shows an attempted transfer of $1,000 from your savings account to your checking account. You would not be happy if the money were successfully withdrawn from your savings account, but never reached your checking account.

To protect against this kind of error, the program that handles your transfer request begins a transaction and then issues the SQL statements needed to move the money from your savings to your checking account. It only commits the transaction if everything succeeds. If a problem occurs, the program instructs the server to undo all the changes made since the transaction began.

Oracle Internal & Oracle Academy Use Only

AUTOCOMMIT and Transaction Control Statements

By default, MySQL runs with the AUTOCOMMIT mode enabled:

- As soon as you execute a statement that modifies a table, MySQL stores the update on disk
- Each statement is treated like a transaction

Use the following statements to control a multiple-statement transaction:

- START TRANSACTION (or BEGIN)
- COMMIT
- ROLLBACK

31

When you issue a START TRANSACTION statement, AUTOCOMMIT mode is disabled. Results of statements after the START TRANSACTION statement that modify a table are not written to the disk. The transaction remains open until you close it with a COMMIT, a ROLLBACK, or any statement that implicitly closes a transaction.

You can disable AUTOCOMMiT. After disabling AUTOCOMMIT mode, changes to transaction-safe tables (such as InnoDB) are not made permanent immediately. You must use a COMMIT statement to store your changes to disk or ROLLBACK statement to ignore the changes. The AUTOCOMMIT mode then reverts to its previous state.

To disable AUTOCOMMIT, enter the following statement:

```
SET AUTOCOMMIT=0;
```

Oracle Internal & Oracle Academy Use Only

Use the following statements to control a multiple-statement transaction:

START TRANSACTION (or BEGIN): Explicitly begins a new transaction and disables the AUTOCOMMIT mode

COMMIT: Makes the changes from the current transaction permanent

ROLLBACK: Cancels the changes from the current transaction

Committing Data in a Transaction

Make the changes to the data and commit the transaction:

```
START TRANSACTION;

INSERT INTO retired_employees (employee_id, first_name, last_name, email, retired_date, job_id, salary, manager_id,
department_id)
SELECT employee_id, first_name, last_name, email, CURDATE(), job_id, salary, manager_id, department_id
FROM employees
WHERE employee_id = 113;

DELETE from employees where employee_id = 113;

COMMIT;
```

32

In the example in the slide, an employee is being moved from the `employees` table to the `retired_employees` table. Making this a transaction ensures that the employee can be added to the `retired_employees` table before being deleted from the `employees` table. If either of these actions fails, the transaction is rolled back. Because both statements execute correctly, the changes are saved by issuing the `COMMIT` statement. In this example, the columns to be inserted must be listed because not all of the columns from the `employees` table are used in the `retired_employees` table. Also, the `retired_employees` table has a `retired_date` column, which is set to the current date in this example, but could be set to a specified date if desired.

Oracle Internal & Oracle Academy Use Only

Rolling Back Changes

Make changes to data and then roll back the changes:

```
START TRANSACTION;

INSERT INTO departments
VALUES (200, 'Operations', NULL, 1700);

UPDATE employees
SET department_id = 200
WHERE department_id =
(SELECT department_id FROM departments WHERE department_name = 'IT');

ROLLBACK;
```

33

In the example in the slide, employees from the IT department are being transferred to the newly created Operations department (200). Creating a transaction ensures that the new Operations department is successfully added to the `departments` table before employees from the IT department have their department ID set to the new department. In this example, the `ROLLBACK` statement returns the departments table and the employees table back to the status they were in before the transaction began. That is, the department that was inserted is removed, and the employees from IT still have their previous department ID.

Oracle Internal & Oracle Academy Use Only

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.

```
START TRANSACTION;

INSERT INTO copy_emp
VALUES (113, 'Luis', 'Popp', 'LPOPP', '515.124.4567',
        STR_TO_DATE('07-12-2015', '%d-%m-%Y'), 'FI_ACCOUNT', 6900, NULL,
        205, 110);

SAVEPOINT first_insert;

INSERT INTO copy_emp
VALUES (175, 'Alyssa', 'Hutton', 'AHUTTON', '011.44.1644.429266',
        STR_TO_DATE('19-03-2013', '%d-%m-%Y'), 'SA_REP', 8800, .25, 149,
        80);

ROLLBACK TO first_insert;

COMMIT;
```

34

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes back to that marker by using the `ROLLBACK TO SAVEPOINT` statement. The effect of the example in this slide is that the first `INSERT` statement is committed, but the second insert statement is rolled back. That is, employee 113 is inserted, but employee 175 is not.

Note that if you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

The following transaction would commit the first and third `INSERT` statements (employees 177 and 179) but not the second one (employee 178):

```
START TRANSACTION;
INSERT INTO copy_emp
        VALUES (177, 'Jack', 'Livingston', 'JLIVINGS', '011.44.1644.429264',
        STR_TO_DATE('23-04-2014', '%d-%m-%Y'), 'SA_REP', 8400, .20, 149, 80);
SAVEPOINT first_insert;
INSERT INTO copy_emp
       VALUES (178, 'Kimberely', 'Grant', 'KGRANT', '011.44.1644.429263',
       STR_TO_DATE('24-05-2015', '%d-%m-%Y'), 'SA_REP', 7000, .15, 149, NULL);
ROLLBACK TO first_insert;
INSERT INTO copy_emp
       VALUES (179, 'Charles', 'Johnson', 'CJOHNSON', '011.44.1644.429262',
       STR_TO_DATE('04-01-2016', '%d-%m-%Y'), 'SA_REP', 6200, .10, 149, 80);
COMMIT;
```

Oracle Internal & Oracle Academy Use Only

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`
- Consistent reads
- Manual Data Locking
  - `FOR UPDATE` clause in a `SELECT` statement
  - `LOCK TABLE` statement

35

Oracle Internal & Oracle Academy Use Only

Consistent Reads

In busy database systems, some transactions are reading data while other transactions are changing that same data.

- A read operation does not place any locks on the tables it is reading. Other transactions are not blocked from modifying the data.
- A read operation reads a snapshot of the data. Other transactions can then modify the data.
- The timing of the snapshot depends on the isolation level.
- The default isolation level for the MySQL InnoDB storage engine is `REPEATABLE READ`.

36

MySQL supports other isolation levels. Those are covered in the MySQL Performance Tuning course.

The default isolation level for the MySQL InnoDB storage engine is `REPEATABLE READ`. With that isolation level, a snapshot is taken with the first read statement of a transaction, and all other reads within that transaction use the same snapshot.

Oracle Internal & Oracle Academy Use Only

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

37

Oracle Internal & Oracle Academy Use Only

FOR UPDATE Clause in a SELECT Statement

- The rows in the EMPLOYEES table where job_id is SA_REP are locked.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
ORDER BY employee_id
FOR UPDATE;
```

- This works only after START TRANSACTION or with AUTOCOMMIT set to 0.
- 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.
- In MySQL, the FOR UPDATE clause must be the last clause in the SELECT statement.

38

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. MySQL offers the FOR UPDATE clause of the SELECT statement to perform this locking. This works only within a transaction (after a START TRANSACTION statement) or if AUTOCOMMIT is set to 0 (off) rather than the default value of 1 (ON).

When you issue a SELECT...FOR UPDATE statement in a transaction, the database 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 server not to wait if the table has been locked by another user. In this case, the query returns an error, and control is returned immediately so that you can perform other work, or you can wait for a period of time before trying again. Without the NOWAIT clause, your process blocks until the table is available, when the locks are released by the other user through the other user's issue of a COMMIT or a ROLLBACK statement.

Oracle Internal & Oracle Academy Use Only

FOR UPDATE Clause: Examples

- You can use the FOR UPDATE clause in a SELECT statement against multiple tables.

```
SELECT e.employee_id, e.last_name, e.job_id
FROM employees e JOIN departments d
USING (department_id)
WHERE d.department_name = 'Shipping'
ORDER BY e.employee_id
FOR UPDATE;
```

- Rows from both the EMPLOYEES and DEPARTMENTS tables are locked.
- Use FOR UPDATE OF *table_name* to qualify the table that you intend to change; then only the rows from that specific table are locked.

39

In the example in the slide, the statement locks rows in the employees table with department_id of 50 (Shipping) and locks rows in the departments table with departments named Shipping.

In MySQL, you can use the FOR UPDATE OF table_*name* to qualify the table that you intend to change. When OF *table_name* is omitted, the query locks all tables referenced by the FROM clause. If an alias is specified as the table name in the SELECT statement, the FOR UPDATE OF clause must use the alias.

The following statements lock only those rows in the employees table with department ID of 50 (Shipping). No rows are locked in the departments table:

```
START TRANSACTION;

SELECT e.employee_id, e.last_name, e.job_id

FROM employees e JOIN departments d

USING (department_id)

WHERE d.department_name = 'Shipping'

ORDER BY e.employee_id

FOR UPDATE OF e;

COMMIT;
```

You can manually lock and unlock tables with the LOCK TABLES and UNLOCK TABLES statements. These statements are covered in the MySQL Performance Tuning course.

Oracle Internal & Oracle Academy Use Only

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

40

In this lesson, you should have learned how to manipulate data in the database by using the `INSERT`, `UPDATE`, `DELETE`, and `TRUNCATE` statements, as well as how to control data changes by using the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements. You should have also learned how to use the `FOR UPDATE` clause of the `SELECT` statement to lock rows for your changes only.

Remember that the server guarantees a consistent view of data at all times.

Oracle Internal & Oracle Academy Use Only

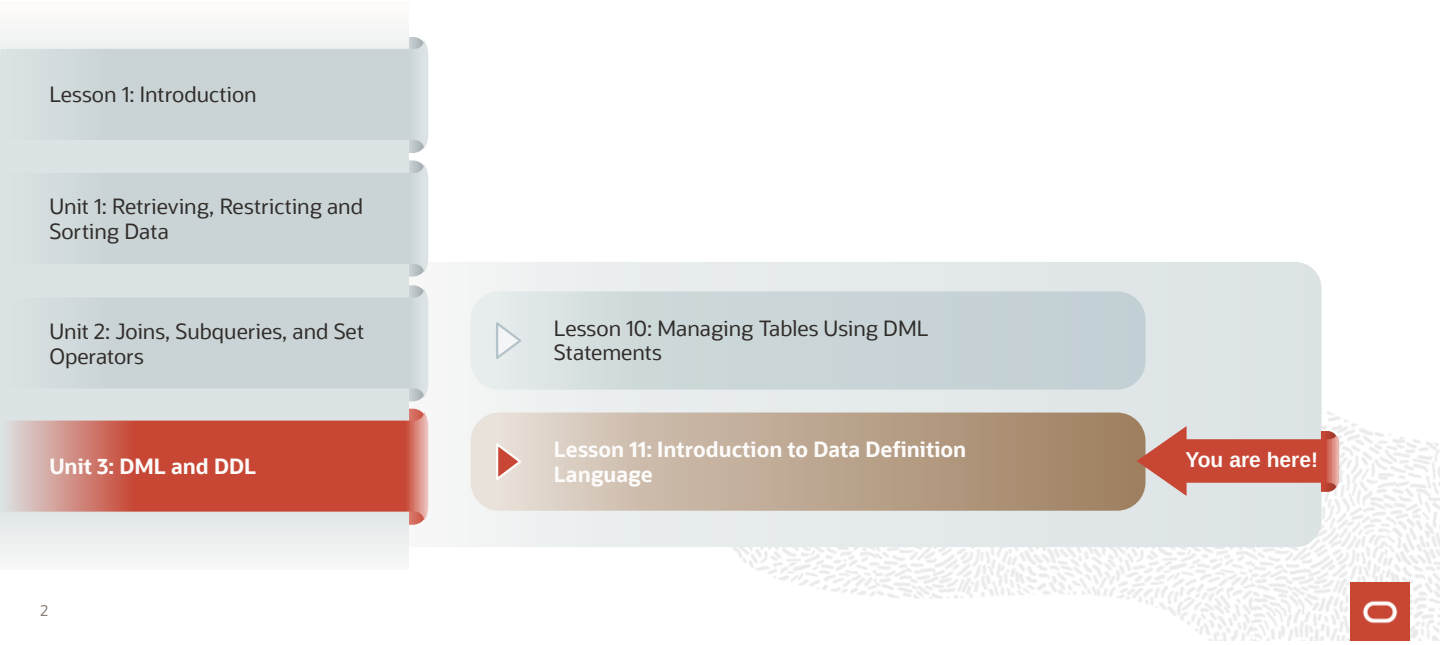
ORACLE

11A

Introduction to Data Definition Language in Oracle

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 3, you learn how to create and manage database objects using data definition language (DDL) statements. You will also learn how to manage data in the tables using data manipulation language (DML) statements.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

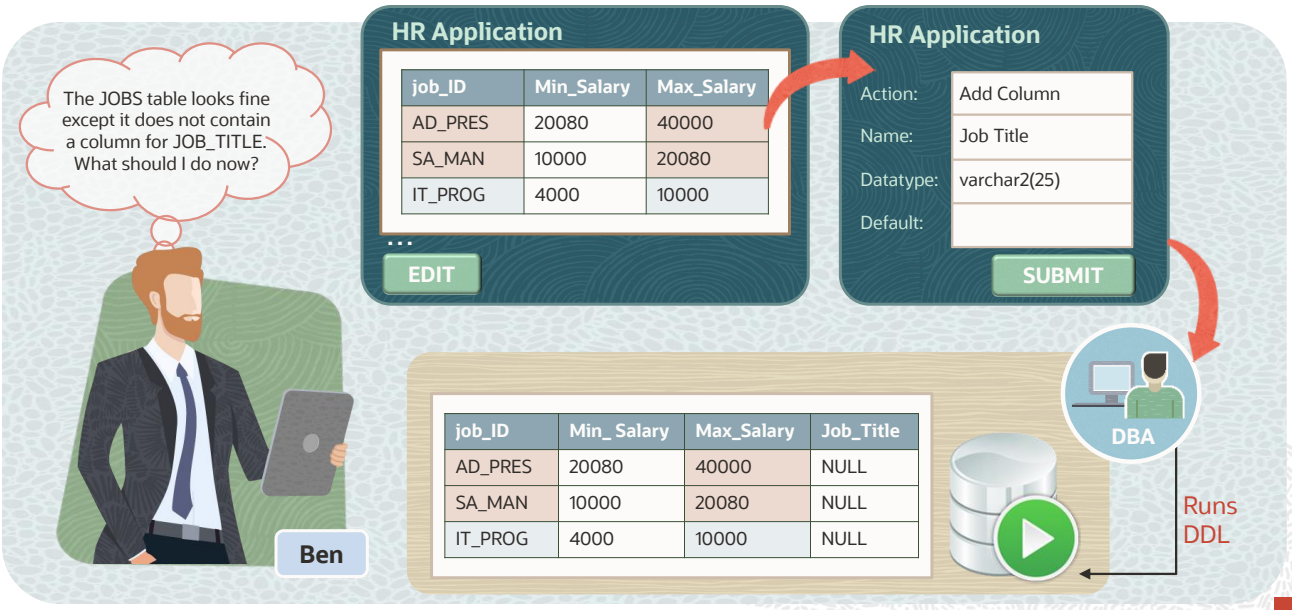
- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation

3

In this lesson, you are introduced to the data definition language (DDL) statements. You learn the basics of creating simple tables, altering them, and removing them. The data types available in DDL are shown and schema concepts are introduced. Constraints are discussed in this lesson. Exception messages that are generated from violating constraints during DML operations are shown and explained.

Oracle Internal & Oracle Academy Use Only

HR Application Scenario



4

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.

Oracle Internal & Oracle Academy Use Only

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

5

Oracle Internal & Oracle Academy Use Only

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

6

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.

Oracle Internal & Oracle Academy Use Only

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

7

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 19c database.

Oracle Internal & Oracle Academy Use Only

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

8

Oracle Internal & Oracle Academy Use Only

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

9

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:

schema	Is the same as the owner's name
table	Is the name of the table
DEFAULT *expr*	Specifies a default value 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

Note: The CREATE ANY TABLE privilege is needed to create a table in any schema other than the user's schema.

Oracle Internal & Oracle Academy Use Only

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
```

10

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 lesson titled “Manipulating Data Using Advanced Queries” in the course.

Oracle Internal & Oracle Academy Use Only

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

11

Oracle Internal & Oracle Academy Use Only

Data Types

Data Type	Description
VARCHAR2(size)	Variable-length character data
CHAR(size)	Fixed-length character data
NUMBER(p, s)	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

12

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(*size*)	Variable-length character data (A maximum *size* must be specified; minimum *size* is 1.) Maximum size is: • 32767 bytes if MAX_SQL_STRING_SIZE = EXTENDED • 4000 bytes if MAX_SQL_STRING_SIZE = LEGACY
CHAR [(*size*)]	Fixed-length character data of length *size* bytes (Default and minimum *size* is 1; maximum *size* is 2,000.)
NUMBER [(*p*,*s*)]	Number having precision *p* and scale *s* (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.

Oracle Internal & Oracle Academy Use Only

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 *size* bytes. You must specify *size* for a RAW value. Maximum *size* is: 32767 bytes if MAX_SQL_STRING_SIZE = EXTENDED 4000 bytes if MAX_SQL_STRING_SIZE = 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.

Oracle Internal & Oracle Academy Use Only

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

14

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: 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 19c database.

Oracle Internal & Oracle Academy Use Only

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.
```

15

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

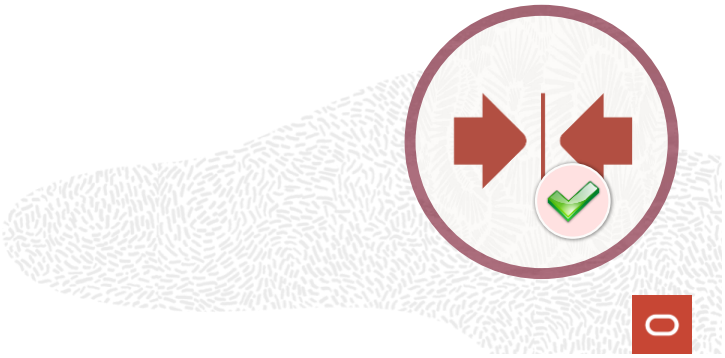
- Database objects
  - Naming rules
- `CREATE TABLE` statement
- Data types
- Overview of constraints: `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK` constraints
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

16

Oracle Internal & Oracle Academy Use Only

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



17

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

Oracle Internal & Oracle Academy Use Only

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.

18

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 19c database.

Oracle Internal & Oracle Academy Use Only

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, ...),
```

19

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

Oracle Internal & Oracle Academy Use Only

Defining Constraints: Example

- Example of a column-level constraint:

1

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

- Example of a table-level constraint:

2

```
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));
```

20

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.

1. The first example uses the column-level syntax to define the constraint.
2. 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.

Oracle Internal & Oracle Academy Use Only

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

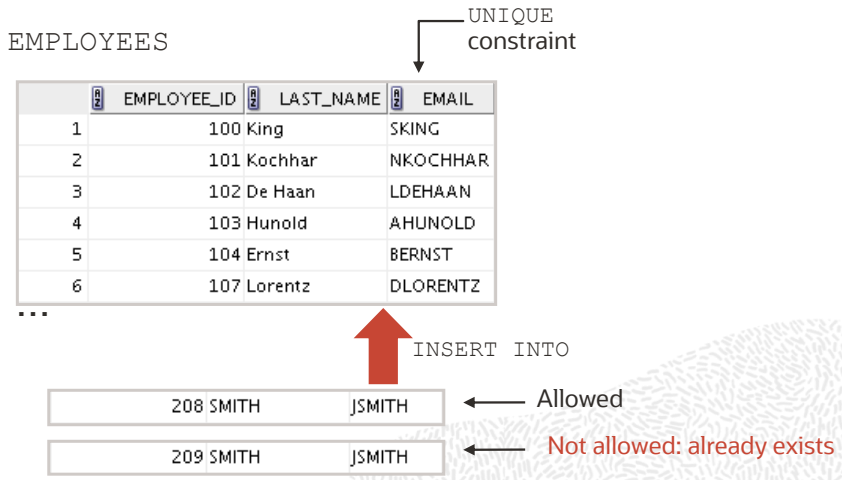
21

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level.

In the EMPLOYEES table, the EMPLOYEE_ID column inherits a NOT NULL constraint because it is defined as a primary key. Otherwise, the LAST_NAME, EMAIL, HIRE_DATE, and JOB_ID columns have the NOT NULL constraint enforced on them.

Oracle Internal & Oracle Academy Use Only

UNIQUE Constraint



22

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.

Oracle Internal & Oracle Academy Use Only

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));
```

23

You can define UNIQUE constraints at the column level or table level.

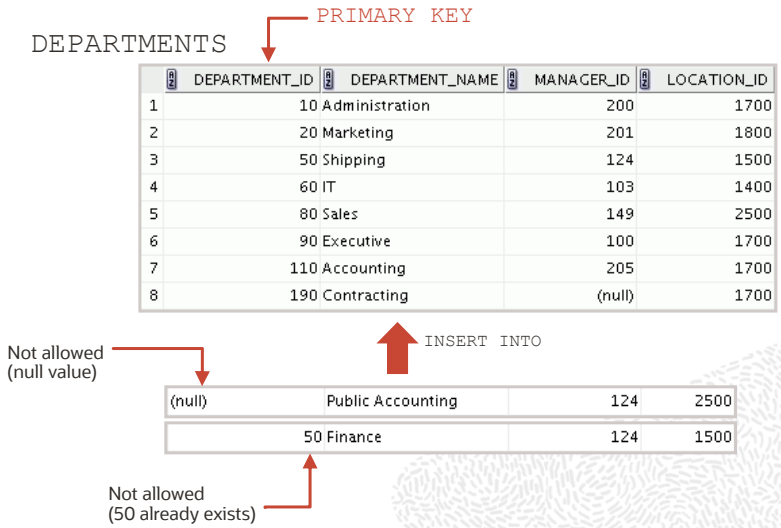
You define the constraint at the table level when you want to create a composite unique key. A composite key is defined when there is not a single attribute that can uniquely identify a row. In that case, you can have a unique key that is composed of two or more columns, the combined value of which is always unique and can identify rows.

The example in the slide applies the UNIQUE constraint to the EMAIL column of the EMPLOYEES table. The name of the constraint is EMP_EMAIL_UK.

Note: The Oracle server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.

Oracle Internal & Oracle Academy Use Only

PRIMARY KEY Constraint



	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

(null)	Public Accounting	124	2500
50	Finance	124	1500

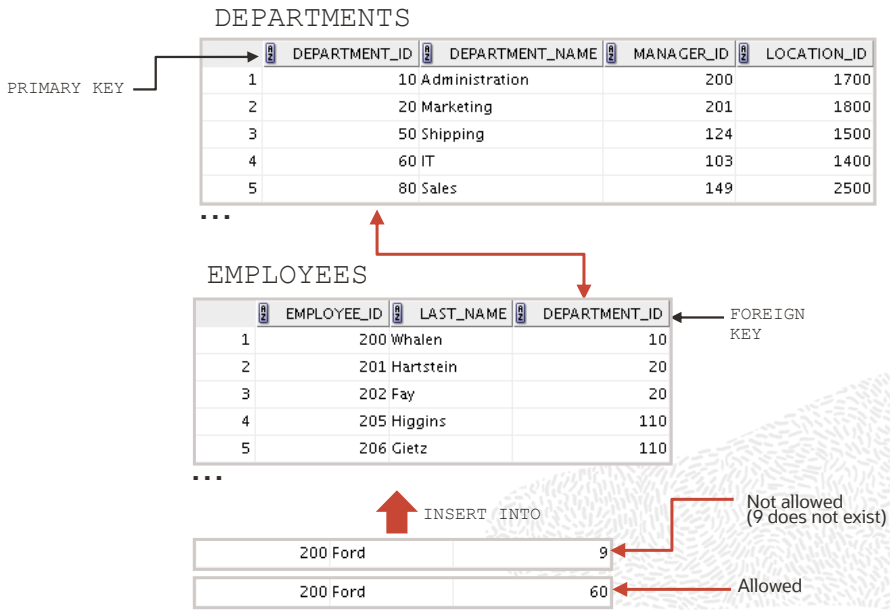
24

A PRIMARY KEY constraint creates a primary key for the table. You can create only one primary key for each table. The PRIMARY KEY constraint is a column or a set of columns that uniquely identifies each row in a table. This constraint enforces the uniqueness of the column or column combination and ensures that no column can contain a null value.

Note: Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.

Oracle Internal & Oracle Academy Use Only

FOREIGN KEY Constraint



25

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.

Oracle Internal & Oracle Academy Use Only

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));
```

26

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),
...
)
```

Oracle Internal & Oracle Academy Use Only

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

27

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.

Oracle Internal & Oracle Academy Use Only

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),...
```

28

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),
   ...
```

Oracle Internal & Oracle Academy Use Only

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));
```

29

The example in the slide shows the statement that is used to create the `TEACH_EMP` table.

Oracle Internal & Oracle Academy Use Only

Violating Constraints

Department 55 does not exist.

```
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 -
ORA-02291: integrity constraint (ORA1.EMP_DEPT_FK) violated - parent key not found
```

30

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`.

Oracle Internal & Oracle Academy Use Only

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 -
ORA-02292: integrity constraint (ORA1.EMP_DEPT_FK) violated - child record found
```

31

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example in the slide tries to delete department 60 from the `DEPARTMENTS` table, but it results in an error because that department number is used as a foreign key in the `EMPLOYEES` table. If the parent record that you attempt to delete has child records, you receive the "child record found" violation `ORA-02292`.

The following statement works because there are no employees in department 70:

```
DELETE FROM  departments
WHERE department_id = 70;
```

```
1 row deleted.
```

Oracle Internal & Oracle Academy Use Only

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

32

Oracle Internal & Oracle Academy Use Only

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.

33

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:

table	Is the name of the table
column	Is the name of the column, default value, and integrity constraint
subquery	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.

Oracle Internal & Oracle Academy Use Only

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
```

34

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:
```

Oracle Internal & Oracle Academy Use Only

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

35

Oracle Internal & Oracle Academy Use Only

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

36

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.

Oracle Internal & Oracle Academy Use Only

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] …);
```

37

You can add columns to a table, modify columns, and drop columns from a table by using the ALTER TABLE statement.

In the syntax:

table	Is the name of the table
ADD|MODIFY|DROP	Is the type of modification
column	Is the name of the column
datatype	Is the data type and length of the column
DEFAULT *expr*	Specifies the default value for a column

Oracle Internal & Oracle Academy Use Only

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)

38

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.

Oracle Internal & Oracle Academy Use Only

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.

39

You can modify a column definition by using the `ALTER TABLE` statement with the `MODIFY` clause. Column modification can include changes to a column's data type, size, and default value.

Guidelines

- You can increase the width or precision of a numeric column.
- You can increase the width of character columns.
- You can decrease the width of a column if:
  - The column contains only null values
  - The table has no rows
  - The decrease in column width is not less than the existing values in that column
- You can change the data type if the column contains only null values. The exception to this is `CHAR`-to-`VARCHAR2` conversions, which can be done with data in the columns.
- You can convert a `CHAR` column to the `VARCHAR2` data type or convert a `VARCHAR2` column to the `CHAR` data type only if the column contains null values or if you do not change the size.
- A change to the default value of a column affects only subsequent insertions to the table.

Oracle Internal & Oracle Academy Use Only

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	Zlotkey	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

40

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.*

Oracle Internal & Oracle Academy Use Only

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;
```

41

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.

Oracle Internal & Oracle Academy Use Only

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.

Oracle Internal & Oracle Academy Use Only

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;
```

43

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.

Oracle Internal & Oracle Academy Use Only

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

44

Oracle Internal & Oracle Academy Use Only

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.
```

45

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.

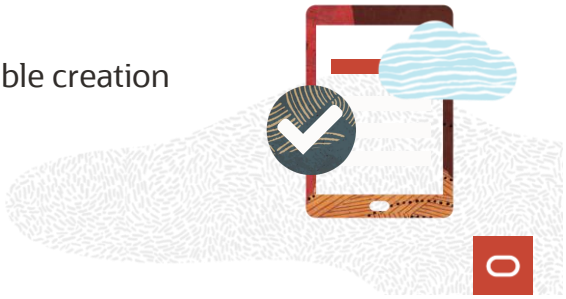
Note: Use the FLASHBACK TABLE statement to restore a dropped table from the recycle bin. This is discussed in detail in the course titled *Oracle Database 19c: SQL Workshop II*.

Oracle Internal & Oracle Academy Use Only

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



46

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.

Oracle Internal & Oracle Academy Use Only

Practice 11a: 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

47

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.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

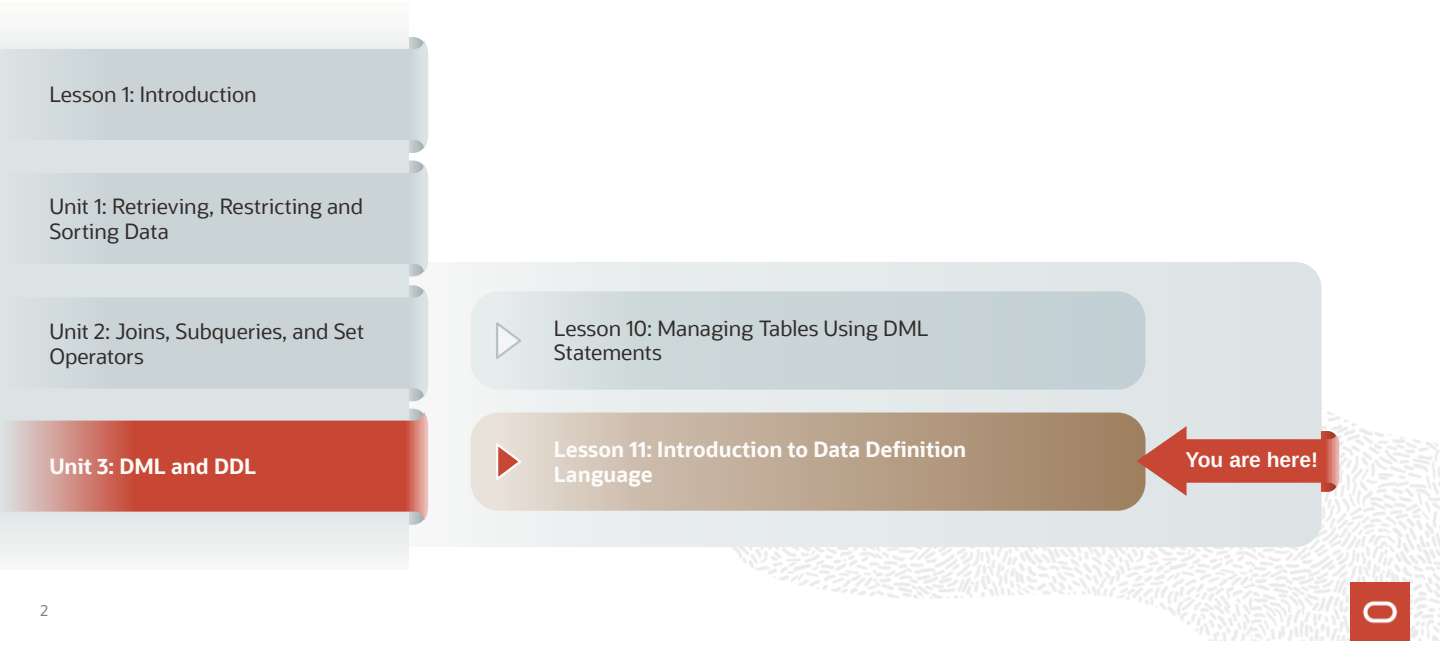
ORACLE

11B

Introduction to Data Definition Language in MySQL

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 3, you learn how to create and manage database objects using data definition language (DDL) statements. You will also learn how to manage data in the tables using data manipulation language (DML) statements.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

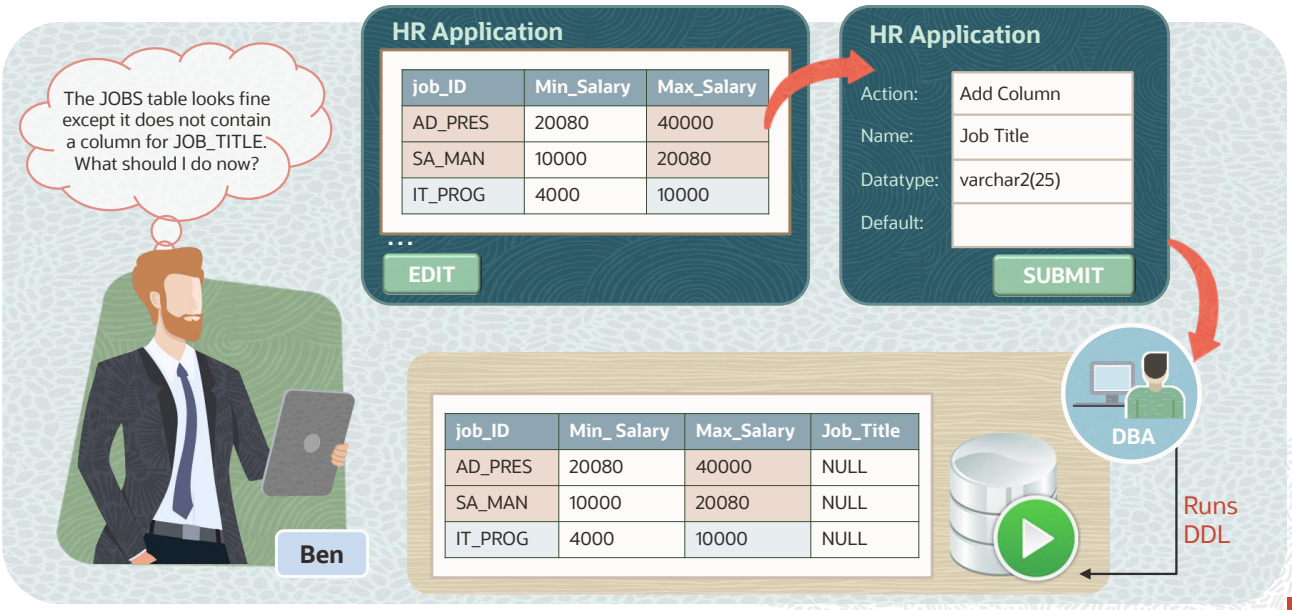
- Create a database
- Add tables to a database
- List the data types that are available for columns
- Use the `SHOW CREATE TABLE` statement
- Describe and set column and table options
- Create indexes, keys, and constraints

3

In this lesson, you are introduced to the data definition language (DDL) statements. You learn the basics of creating simple tables, altering them, and removing them. The data types available in DDL are shown and schema concepts are introduced. Constraints are discussed in this lesson. Exception messages that are generated from violating constraints during DML operations are shown and explained.

Oracle Internal & Oracle Academy Use Only

HR Application Scenario



4

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.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

5

Oracle Internal & Oracle Academy Use Only

Creating a Database: Syntax

General syntax for creating a database:

```
CREATE DATABASE [IF NOT EXISTS] database_name;
```

Or

```
CREATE SCHEMA [IF NOT EXISTS] database_name;
```

The IF NOT EXISTS option prevents getting an error if the database already exists.

6

The terms database and schema are synonymous in MySQL. In this course, you have been working with one database or schema, the HR database, with tables for employees and jobs. You might also have an inventory database with tables for items, their locations, manufacturers, costs, and so on.

The statement to create the HR database is the following:

```
CREATE DATABASE IF NOT EXISTS HR;
```

Oracle Internal & Oracle Academy Use Only

MySQL Naming Conventions

Database names, table names, and column names:

- Can contain A–Z, a–z, 0–9, _, and $
- Cannot have more than 64 characters
- Cannot contain reserved words or special characters, such as /, \, ., # or spaces unless you enclose the name with backticks

For example, the following causes an error:

```
CREATE DATABASE my database;
```

Correct the error by enclosing the name in backticks:

```
CREATE DATABASE `my database`;
```

7

Databases and tables in MySQL are created by using directories and files in the host file system. Because of this, if the host operating system is case-sensitive in its treatment of directory and file names, so is MySQL. Windows file names are not case-sensitive, so a server running on Windows does not treat database and table names as case-sensitive. MySQL servers running on UNIX usually treat database and table names as case-sensitive because UNIX file names are case-sensitive. Case sensitivity of database and table names depends on the host operating system.

Table names can be reserved words or contain special characters if the name is quoted in backticks (``).

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

8

Oracle Internal & Oracle Academy Use Only

CREATE TABLE Statement

General syntax for creating a table:

```
CREATE TABLE [database.]table
          (column_name datatype [, ...]);
```

- Example:

```
CREATE TABLE employees1
          (employee_id    INTEGER,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25),
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER);
```

9

You create tables to store data by executing the SQL `CREATE TABLE` statement. This statement is one of the DDL statements that are used to create, modify, or remove database structures. These statements 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. The example in this slide creates a table similar to the employees table It creates a table with columns and their datatypes specified. This lesson describes numerous other options that can be used when creating tables. The table in this slide uses the data types in the employee table. MySQL supports many other data types.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

10

Oracle Internal & Oracle Academy Use Only

Data Types: Overview

- Numeric data types:
  - Integer, decimal, and floating point data
- Date and time data types:
  - Calendar dates, with or without time data and time intervals
- String data types:
  - Character strings and BLOBs
- Spatial data types:
  - Data representing geographic features
- JavaScript Object Notation (JSON):
  - Structured text data stored in JSON documents

11

MySQL supports a wide variety of data types. In this course, you have worked with examples of numeric, date, and character data. This lesson describes some of the other commonly used MySQL data types. Spatial data and JSON documents are beyond the scope of this course. Data types are covered in more depth in the MySQL Essentials course.

Oracle Internal & Oracle Academy Use Only

Numeric Data Types

Integer:

- Positive or negative whole numbers
- Syntax: `INT[(M)] [UNSIGNED] [ZEROFILL]`

Fixed Point:

- Used for exact-value numbers: integer, fractional, or both
- Syntax: `DECIMAL[(M[, D])] [UNSIGNED] [ZEROFILL]`

Floating Point:

- Used for approximate-value numbers: integer, fractional, or both
- Syntax: `FLOAT[(M, D)] [UNSIGNED] [ZEROFILL]`

12

In the syntax, *M* represents an optional value for the maximum number of digits available, and, for `DECIMAL` and `FLOAT`, *D* represents the number of digits following the decimal point. `INTEGER` is a synonym for `INT`. For `INT`, if *M* is omitted, a default number of digits is provided.

`NUMERIC` and `DEC` are synonyms for `DECIMAL`. For `DECIMAL` data type, if *D* is omitted, the default is 0 decimal places, and if *M* is omitted, the default is 10 significant digits. The decimal point and (for negative numbers) the minus (-) sign are not counted in *M*.

Floating point numbers are used to represent very large or very small numbers (close to zero) using a small amount of space. For example, `6.022169E+23` represents `602216900000000000000000`, while `6.022169E-23` represents `.00000000000000006022169` as a value. If *M* and *D* are omitted, `FLOAT` values are stored to the limits permitted by the hardware.

MySQL supports a number of other integer, fixed point, and floating point data types for efficiency of storage or calculations. Other data types are covered in the course on MySQL Essentials.

Oracle Internal & Oracle Academy Use Only

Date and Time Data Types

DATE:

- YYYY-MM-DD as in 2018-01-04

TIME:

- HH:MM:SS as in 12:59:02

DATETIME:

- YYYY-MM-DD HH:MM:SS as in 2018-01-04 12:59:02

13

Here, YYYY, MM, DD, HH, MM, and SS stand for year, month, day of month, hour, minute, and second, respectively.

You have used the DATE data type previously in this course.

The TIME data type can express time of day or elapsed time.

The DATETIME data type is a combination of date and time.

The TIME and DATETIME data types can include fractional seconds up to 6 decimal places.

Other date and time data types and the use of fractional seconds are covered in the course on MySQL Essentials.

Oracle Internal & Oracle Academy Use Only

String Data Types

- Fixed length character strings: `CHAR` (*M*)
  - Stored as fixed length *M*, right-padded with spaces
- Variable length character strings: `VARCHAR` (*M*)
  - Stored **up to** length *M*, **not** right-padded with spaces
- Extremely long character strings: `TEXT`
  - Variable length character strings, stored as a separately allocated object
- **B**inary **l**arge **ob**jects: `BLOB`
  - Variable length binary data, stored as a separately allocated object

14

The `CHAR` data type is efficient for storage when the character string column always has the same number of characters, such as for a 2-character state or region abbreviation or a 3-character country abbreviation. The `VARCHAR` data type is common for character strings that are variable in length, such as names of people or places. For `CHAR`, length can be up to 255, while for `VARCHAR`, length can be up to 65,535, but cannot cause the size of the row, including all columns, to exceed 65,535 bytes. For extremely long character strings, the `TEXT` type can hold a variable amount of data. A `BLOB` (binary large object) is binary data stored as a single value in a database.

The course on MySQL Essentials provides more coverage of the `TEXT` and `BLOB` data types, along with `BINARY`, `VARBINARY`, `ENUM`, and `SET` data types.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

15

Oracle Internal & Oracle Academy Use Only

Indexes, Keys, and Constraints

- **Index:** A column or columns of table data designated to be used as pointers to table rows to speed up data retrieval
- **Key:** Normally a synonym for index in MySQL
- **Constraint:** A restriction placed on one or more column values of a table to enforce integrity rules

The following MySQL indexes (keys) include designated combinations of constraints:

- Primary Keys
- Unique Keys
- Foreign Keys

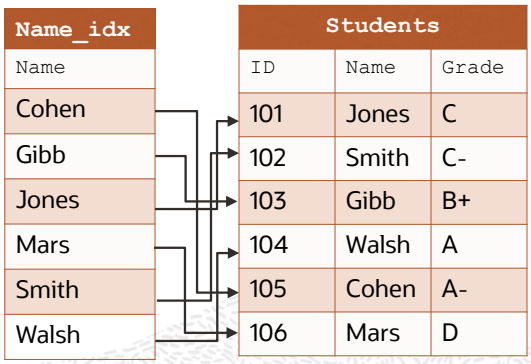
16

A constraint is a condition that data must meet when being added to or removed from a table. For example, a unique constraint requires all values in an index to be distinct. If you try to add a new row with a value in the index that matches a value in an existing row, an error occurs. Primary keys, unique keys, and foreign keys are indexes that have specific combinations of constraints. The sections in this lesson titled "Primary Keys", "Unique Key Constraints", and "Foreign Key Constraints" describe the constraints specific to these indexes and how you can include these types of indexes in a table.

Oracle Internal & Oracle Academy Use Only

Table Indexes

- An index is a collection of pointers to records in a table.
- It helps to:
  - Locate rows quickly
  - Avoid full table scans
  - Improve query performance
- Unnecessary indexes are wasteful:
  - Take up space in the database
  - Updated whenever data is updated



17

By default, when MySQL attempts to locate a record, it scans the entire table until a match is found (full table scan). This can really slow down large queries.

An index helps MySQL find specific column values quickly. The index entries act as pointers to the rows that match these criteria so they can be accessed without a full table scan.

You can assign indexes when you create a table, or at any time afterward. You can create indexes on single columns or multiple columns (composite indexes). For example, you can use the phone number for an individual in a phone book as a single column index, or you can use the last name and first name to create a composite index.

You can index any table column, but do not overdo it. Indexes take up space. Too many indexes can slow performance because indexes are updated whenever you add or delete rows or modify data in indexed columns. Avoid indexing fields that change regularly. Selecting columns for indexing is covered in the MySQL Database Administrators course.

Note: A B-tree is a type of data structure used by DBMSs to store indexes. The data is kept sorted at all times, which makes for fast lookup of exact matches or comparisons of values.

Oracle Internal & Oracle Academy Use Only

Primary Keys

A primary key has the following constraints:

- Uniquely identifies a single row in the table
- Null values not allowed
- Only one primary key allowed per table

```
CREATE TABLE employees2
          (employee_id    INTEGER,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25),
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER,
           PRIMARY KEY (employee_id));
```

18

The primary key is the identifying key for the table. Each row in the table must have a unique value for the primary key and it cannot be NULL. The primary key can be a composite key, which is covered elsewhere in this lesson. The InnoDB engine stores tables sorted by the primary key.

The example in this slide uses the `PRIMARY KEY` clause at the end of the statement. Another option is to use the `PRIMARY KEY` attribute in the column definition for `ID`, as shown in this example:

```
CREATE TABLE employees2
          (employee_id    INTEGER PRIMARY KEY,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25),
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER);
```

Oracle Internal & Oracle Academy Use Only

Unique Key Constraints

Unique keys have the following constraints:

- The value must be unique for that column within the table.
- Null values are allowed for unique keys.
- Multiple unique keys are allowed per table.

```
CREATE TABLE employees3
          (employee_id    INTEGER,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25),
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER,
           PRIMARY KEY (employee_id),
           UNIQUE KEY (email));
```

19

The unique key constraint requires that all values in the column are different. Primary keys are unique keys that have the additional constraint that null values are not allowed and only one primary key is allowed per table. You can specify unique keys that do not have the additional constraints of primary keys. In the example in this slide, each employee's email must be unique. The example in this slide uses `UNIQUE KEY` clauses at the end of the statement. Another option is to use `UNIQUE KEY` attributes in the column definitions, as shown in this example:

```
CREATE TABLE employees3
          (employee_id    INTEGER PRIMARY KEY,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25) UNIQUE KEY,
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER);
```

Oracle Internal & Oracle Academy Use Only

Foreign Key Constraints

Foreign keys maintain referential integrity by enforcing a link between the data in two tables:

- The foreign key in the child table references the primary key in the parent table.
- The foreign key constraint prevents actions that would destroy the links between the child and parent tables.

20

When you add a record to a table with a foreign key constraint, MySQL checks whether the value being added for the foreign key column has a matching value in the column referenced in the parent table and rejects the operation if no match is found. When you delete or update a record in a parent table, MySQL checks whether the value in the primary key column has any matching values in the foreign key column of the child table. Depending on how the foreign key is defined, the operation might be rejected, or the row might be deleted, updated, or set to a specified value.

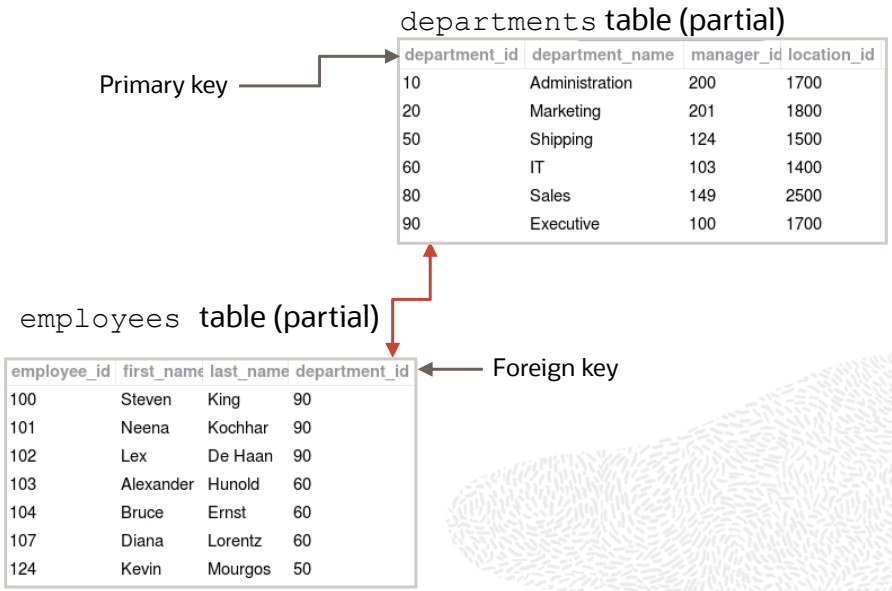
For example:

- When inserting or updating a row in a child table, the value for the foreign key column being inserted or updated must already exist as the primary key for a row in the parent table. Otherwise, the insert or update is rejected.
- When deleting or updating a row in the parent table, the foreign key constraint determines what happens to any rows in the child table with a foreign key matching the column being deleted or updated.

The actions are described in the section titled "`FOREIGN KEY` Constraint: Referential Actions" in this lesson.

Oracle Internal & Oracle Academy Use Only

Foreign Key Constraint: Example Tables



21

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 (child table); it references the `department_id` column of the `departments` table (the parent table). If you try to insert a row into the employees table or change the department_id of a row in the employees table, the value provided for department_id must already be a primary key in the departments table. If you try to delete a row from the departments table or try to change the value for a department_id in the departments table, it will first check whether there is a row in the employees table with that foreign key and take the action defined for the foreign key, described later in this lesson.

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.
- The primary key and foreign key might have the same column name (as in this example), but that is not required and often not appropriate.

Oracle Internal & Oracle Academy Use Only

FOREIGN KEY Constraint: Example Statement

```
CREATE TABLE employees4
          (employee_id    INTEGER,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25),
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER,
           PRIMARY KEY (employee_id),
           UNIQUE KEY (email),
           CONSTRAINT emp4_dept_fk
               FOREIGN KEY (department_id)
               REFERENCES departments (department_id));
```

22

The example in the slide defines a `FOREIGN KEY` constraint on the `department_id` column of the `employees` table. This makes `department_id` in the `employees4` child table a foreign key that must match a value for the primary key `department_id` in the `departments` table. The name of the constraint is `emp4_dept_fk`. The name must be unique for all tables in the database. If you do not provide a name for the foreign key constraint, MySQL will generate a name. The `CONSTRAINT` keyword is optional. If `CONSTRAINT` is omitted, the foreign key name must be omitted.

Oracle Internal & Oracle Academy Use Only

FOREIGN KEY Constraint: Referential Actions

To control what happens to rows in a child table when a DELETE or UPDATE operation affects a key value in a parent table that has matching rows in the child table, you can include one or both of these optional subclauses at the end of the FOREIGN KEY clause:

```
[ON DELETE referential_action]
[ON UPDATE referential_action]
```

The *referential_action* can be one of the following:

- RESTRICT or NO ACTION
- CASCADE
- SET NULL

23

If ON DELETE or ON UPDATE is not included at the end of a FOREIGN KEY clause, the default is to reject a deletion or update on the parent table. This is the same as specifying RESTRICT or NO ACTION. Specifying ON DELETE CASCADE in the example with the department_id foreign key would mean that, if you delete a department from the departments table, all employees with that same department_id would be deleted. Specifying ON UPDATE CASCADE in the example would mean that, if you change the department_id in the department table, all employees with that department_id would be assigned the newly updated department_id. If you specify ON DELETE SET NULL or ON UPDATE SET NULL, be sure the column in the child table is not defined as NOT NULL.

The *referential_action* can be one of the following:

RESTRICT or NO ACTION: This rejects the DELETE or UPDATE operation on the parent table. It is the default action if ON DELETE or ON UPDATE is omitted.

CASCADE: This deletes or updates the row in the parent table and automatically deletes or updates the matching rows in the child table.

SET NULL: This deletes or updates the row in the parent table and sets the foreign key column or columns in the child table to NULL.

Oracle Internal & Oracle Academy Use Only

Secondary Indexes

A secondary index does not have any constraints and is used to speed up access to data. For example:

```
CREATE TABLE employees5
          (employee_id    INTEGER,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25),
           email          VARCHAR(25),
           phone_number   VARCHAR(20),
           hire_date      DATE,
           job_id         VARCHAR(10),
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER,
           department_id  INTEGER,
           PRIMARY KEY (employee_id),
           UNIQUE KEY (email),
           INDEX emp5_name_ix (last_name),
           CONSTRAINT emp5_dept_fk
               FOREIGN KEY (department_id)
               REFERENCES departments (department_id));
```

24

A secondary index (or key) does not have to be unique or have any referential constraints. The index provides faster access to data. For example, if you know that many searches will be done on employees' last names, you can create an index for the `last_name` column. Last names do not have to be unique. Any kind of index, including primary key, unique key, foreign key, or secondary index, can be a composite key combining multiple columns to form the key value. For primary and unique keys, the individual parts of the composite key do not have to be unique, but the value of the combined columns must be unique.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

25

Oracle Internal & Oracle Academy Use Only

Column Options

You can add options to the CREATE TABLE statement's column definitions, including NULL, NOT NULL, DEFAULT, or AUTO_INCREMENT.

Example:

```
CREATE TABLE employees6
          (employee_id    INTEGER AUTO_INCREMENT,
           first_name     VARCHAR(20),
           last_name      VARCHAR(25) NOT NULL,
           email          VARCHAR(25) NOT NULL,
           phone_number   VARCHAR(20),
           hire_date      DATE NOT NULL,
           job_id         VARCHAR(10) NOT NULL,
           salary         DECIMAL(8,2),
           commission_pct DECIMAL(2,2),
           manager_id     INTEGER DEFAULT 1,
           department_id  INTEGER DEFAULT 10,
           PRIMARY KEY (employee_id),
           UNIQUE KEY (email),
           INDEX emp6_name_ix (last_name),
           CONSTRAINT emp6_dept_fk
               FOREIGN KEY (department_id)
               REFERENCES departments
      (department_id));
```

26

The example in this slide creates a table similar to the employees table. You can include NULL as an option or omit it because it is the default and means that the column can accept NULL as a value. Specifying NOT NULL indicates that the column must have a value and cannot be set to NULL. The DEFAULT option specifies the default value to use when a row is inserted. To use the default value, if specifying the list of column names in the INSERT statement, omit the column name for the column to be set to its default and don't provide a value for it in the VALUES clause. Alternatively, you can use the DEFAULT keyword in the VALUES clause to set the column to its default value.

The AUTO_INCREMENT attribute can be set on an indexed column and sets the value for the column to the next higher value. When inserting a new value, either omit the value from the list of column names and VALUES clause or specify the value as NULL, and it will be assigned the next higher value.

The following statement would set the employee_id to the next higher integer value, set manager_id and department_id to their default values of 1 and 10, respectively, and set any columns not listed to NULL.

```
INSERT INTO employees6 (last_name, email, hire_date, job_id)
                VALUES ('jones', 'jones', '2010-01-01', 'AC_MGR');
```

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

27

Oracle Internal & Oracle Academy Use Only

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.

28

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:

- ***table:*** the name of the table
- ***column:*** the name of the column, default value, and integrity constraint
- ***subquery:*** 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.

Oracle Internal & Oracle Academy Use Only

Creating a Table Using a Subquery: Example

```
CREATE TABLE dept80
  AS
   SELECT   employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
   FROM     employees
   WHERE    department_id = 80;
```

```
DESCRIBE dept80;
```

#	Field	Type	Null	Key	Default	Extra
1	employee_id	int(11)	NO		NULL	
2	last_name	varchar(25)	NO		NULL	
3	ANNSAL	decimal(10,2)	YES		NULL	
4	hire_date	date	NO		NULL	

```
SELECT * FROM dept80;
```

#	employee_ic	last_name	ANNSAL	hire_date
1	149	Zlotkey	126000.00	2016-01-29
2	174	Abel	132000.00	2012-05-11
3	176	Taylor	103200.00	2014-03-24

29

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` statement. The employee_id column is not set as the primary key, but the `NOT NULL` attributes are maintained. You can provide a column alias when using an expression in the subquery. In the example, the expression `salary*12` is given the alias `ANNSAL`.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

30

Oracle Internal & Oracle Academy Use Only

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
- Add an index or constraint

31

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.

Oracle Internal & Oracle Academy Use Only

ALTER TABLE Statement: Add, Modify, or Drop Columns

Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD [COLUMN] column datatype [column_options]
        [FIRST | AFTER column_name];
```

```
ALTER TABLE table
MODIFY [COLUMN] column datatype [column_options]
        [FIRST | AFTER column_name];
```

```
ALTER TABLE table
DROP [COLUMN] column;
```

32

You can add columns to a table, modify columns, and drop columns from a table by using the ALTER TABLE statement.

Oracle Internal & Oracle Academy Use Only

Adding a Column

- You use the `ADD` clause to add columns:

```
ALTER TABLE dept80
ADD job_id VARCHAR(10) NOT NULL DEFAULT 'ST_CLERK'
    AFTER last_name;
```

- The new job_id column is added after the `last_name` column, with the values set to `ST_CLERK`:

```
SELECT * FROM dept80;
```

#	employee_id	last_name	job_id	ANNSAL	hire_date
1	149	Zlotkey	ST_CLERK	126000.00	2016-01-29
2	174	Abel	ST_CLERK	132000.00	2012-05-11
3	176	Taylor	ST_CLERK	103200.00	2014-03-24

33

The example in the slide adds a column named `job_id` to the `dept80` table.. Because this table already contains data, in order to set the `NOT NULL` option, you need to provide a default value, and all the rows are provided with that default value in that column. If the table is empty, you can specify the `NOT NULL` option without providing a default value. If you do not specify the `NOT NULL` or `DEFAULT` options, the values of the new column are set to `NULL`. In the example, the `job_id` column is inserted after the `last_name` column If you specify `FIRST` rather than `AFTER`, the column is added before the `employee_id` column. If you do not specify either `FIRST` or `AFTER`, by default, the column is added at the end.

Oracle Internal & Oracle Academy Use Only

Modifying a Column

- You can change a column's data type, size, column options or position.

```
ALTER TABLE dept80
MODIFY last_name VARCHAR(30) NOT NULL FIRST;
```

```
DESCRIBE dept80;
```

#	Field	Type	Null	Key	Default	Extra
1	last_name	varchar(30)	NO		NULL	
2	employee_id	int(11)	NO		NULL	
3	job_id	varchar(10)	NO		ST_CLERK	
4	ANNSAL	decimal(10,2)	YES		NULL	
5	hire_date	date	NO		NULL	

- The size of the `last_name` column is modified, and it is moved to the first column.
- The full column definition must be included. If `NOT NULL` had been omitted, the column would have been changed from `NOT NULL` to accepting `NULL` values.
- If a default value is provided, it affects only subsequent insertions to the table.

34

You can modify a column definition by using the `ALTER TABLE` statement with the `MODIFY` clause. Column modification can include changes to a column's data type, size, or column options like DEFAULT and NOT NULL. You can also change the position of the column with FIRST or AFTER. The column remains in its previous position if you omit FIRST or AFTER.

Oracle Internal & Oracle Academy Use Only

Dropping a Column

Use the DROP COLUMN clause to drop columns that you no longer need from the table:

```
ALTER TABLE  dept80
DROP COLUMN job_id;
```

```
SELECT * FROM dept80;
```

#	last_name	employee_ic	ANNSAL	hire_date
1	Zlotkey	149	126000.00	2016-01-29
2	Abel	174	132000.00	2012-05-11
3	Taylor	176	103200.00	2014-03-24

35

You can drop a column from a table by using the ALTER TABLE statement with the DROP COLUMN clause. If a table includes only one column, the column cannot be dropped. A MySQL extension to standard SQL permits a single ALTER TABLE statement to contain multiple ADD, MODIFY, or DROP clauses, separated by commas. For example, the following statement drops multiple columns:

```
ALTER TABLE dept80
DROP COLUMN ANNSAL,
DROP COLUMN hire_date;
```

Oracle Internal & Oracle Academy Use Only

ALTER TABLE Statement: Add an Index or Constraint

Normally, you create indexes and constraints by using the CREATE TABLE statement when you create the table. Use the ALTER TABLE statement to add an index or constraint:

```
ALTER TABLE table
ADD PRIMARY KEY (column_name);
```

```
ALTER TABLE table
ADD [UNIQUE] INDEX (column_name);
```

```
ALTER TABLE table
ADD CONSTRAINT key_name
FOREIGN KEY (child_column_name)
REFERENCES parent_table (parent_column_name);
```

36

By using an ALTER TABLE statement, you can add a primary key (only if the table does not yet have a primary key, a UNIQUE index, a secondary index, or a foreign key). If you are adding a primary key or unique index to a table that contains data, the statement is rejected if the values are not unique. If you add a foreign key to a table that already contains data, if the values in the foreign key column in the child table do not have corresponding values in the primary key of the parent table, the statement is rejected.

Oracle Internal & Oracle Academy Use Only

ALTER TABLE to Add a Constraint or Index: Example

The following example adds the constraint of manager_id being a foreign key in the employees5 table that references the employee_id in the same table.

```
ALTER TABLE employees6
ADD CONSTRAINT emp6_manager_fk
FOREIGN KEY (manager_id)
REFERENCES employees6 (employee_id);
```

37

The statement in this slide creates a foreign key for the manager_id column in the employees5 table that references the employee_id in the same table. Any manager_id must be the employee_id of some employee in the table.

Oracle Internal & Oracle Academy Use Only

Creating Indexes by Using the CREATE INDEX Statement

With the CREATE INDEX statement, you can add a unique key or a secondary index with no constraints to a table.

Syntax:

```
CREATE [UNIQUE] INDEX index_name
 ON table_name (column_name);
```

Example:

```
CREATE INDEX emp6_job_ix ON employees6 (job_id);
```

38

The example in this slide creates a secondary index for job_id in the employees6 table. If you find that many queries use job_id as a search criteria, you might find it useful to add this index to speed up searches. Note that job_id in the employees6 table cannot be a unique key because many people might have the same job_id. This statement is a shorter version of the following ALTER TABLE statement:

```
ALTER TABLE employees6

ADD INDEX emp6_job_ix (job_id);
```

Oracle Internal & Oracle Academy Use Only

Viewing Index Definitions by Using the `SHOW INDEX` Statement

General syntax for viewing index definitions:

```
SHOW INDEX FROM table_name [FROM database_name]
```

Example:

```
SHOW INDEX FROM employees6;
```

#	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible
1	employees6	0	PRIMARY	1	employee_id	A	1	NULL	NULL		BTREE			YES
2	employees6	0	email	1	email	A	1	NULL	NULL		BTREE			YES
3	employees6	1	emp6_name_ix	1	last_name	A	1	NULL	NULL		BTREE			YES
4	employees6	1	emp6_dept_fk	1	department_id	A	1	NULL	NULL	YES	BTREE			YES
5	employees6	1	emp6_manager_fk	1	manager_id	A	1	NULL	NULL	YES	BTREE			YES
6	employees6	1	emp6_job_ix	1	job_id	A	1	NULL	NULL		BTREE			YES

39

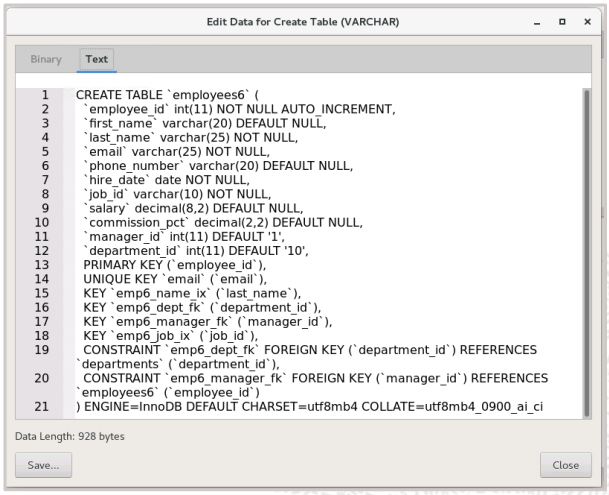
If the database containing the table has been set as the default, you can omit the database name from the statement. In the output, the `Non_unique` column displays a 1 if the index does not have the unique constraint and 0 if it is unique. The output displays the name of the index as well as the columns being indexed. The primary key always has the name `PRIMARY`. The Null column indicates if the column can be set to `NULL`.

Oracle Internal & Oracle Academy Use Only

Showing How a Table Was Created with the `SHOW CREATE TABLE` Statement

View the statement that can be used to create the table. For example:

```
SHOW CREATE TABLE employees6;
```



40

The `SHOW CREATE TABLE` statement produces a statement that can create a table with the same properties as the table specified. This helps you to understand the structure of a table. You can use it to create a new table with the same structure. It does not show the exact statement that was used to create the table. It includes any modifications to the table from `ALTER` statements and might include more information, for example, about default values, than were specified in any `CREATE TABLE` or `ALTER TABLE` statements. For example, the output shows default values of `NULL` for some columns even though that was not specified in any statement. The CREATE TABLE statement also includes some table options at the end even though those were not specified when creating the table.

Note: The results of the statement in MySQL Workbench might be too long to read in the Result Grid. Right-click the output and select **Open Value in Viewer**. Select the **Text** tab. You can save the generated CREATE TABLE statement to a file. When you are finished, click the **Close** button.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Database objects
  - Naming rules
- **`CREATE TABLE` statement**
- Data types
- Indexes, keys, and constraints
- Column options
- Creating a table using a subquery
- `ALTER TABLE` statement
- `DROP TABLE` statement

41

Oracle Internal & Oracle Academy Use Only

Dropping a Table

`DROP TABLE` removes the table definition and all its data entirely.
Syntax:

```
DROP TABLE [IF EXISTS] table_name [, table_name] ...;
```

The optional `IF EXISTS` clause prevents an error from occurring if the table does not exist. If it is omitted, dropping a table that does not exist generates an error.
Example:

```
DROP TABLE IF EXISTS dept80;
```

42

You can drop multiple tables in one statement. For example:

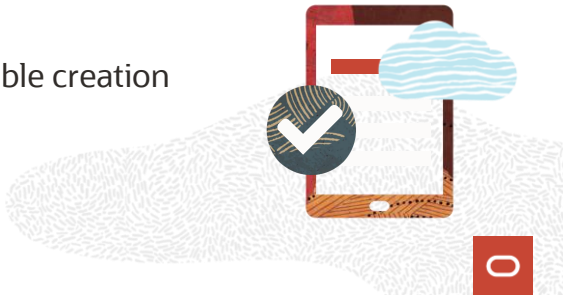
```
DROP TABLE IF EXISTS
employees1, employees2, employees3, employees4, employees5;
```

Oracle Internal & Oracle Academy Use Only

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



43

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.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only