ORACLE
University

Oracle Database 19c: SQL Workshop

Student Guide – Volume II

D108644GC10

Offered in the Oracle Academy Lesson 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

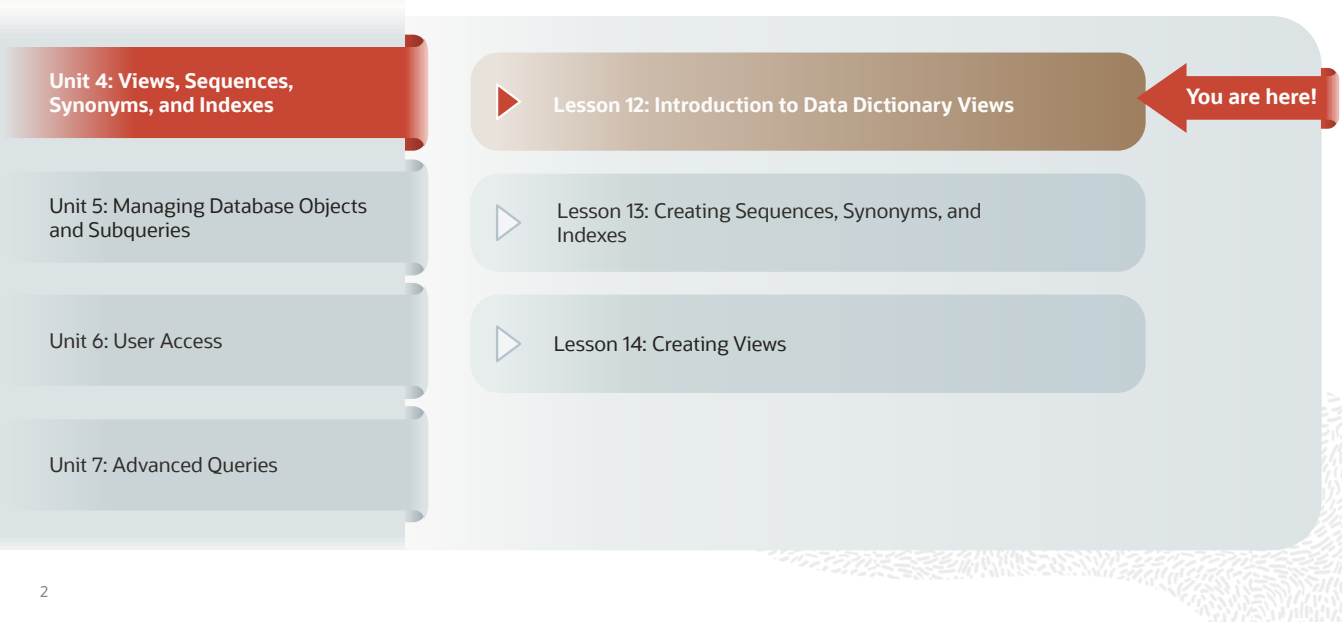
ORACLE

12

Introduction to Data Dictionary Views

Oracle Internal & Oracle Academy Use Only

Course Roadmap



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

Objectives

After completing this lesson, you should be able to:

- Use data dictionary views to research data on your objects
- Query various data dictionary views

3

In this lesson, you are introduced to data dictionary views. You learn that dictionary views can be used to retrieve metadata and create reports about your schema objects.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

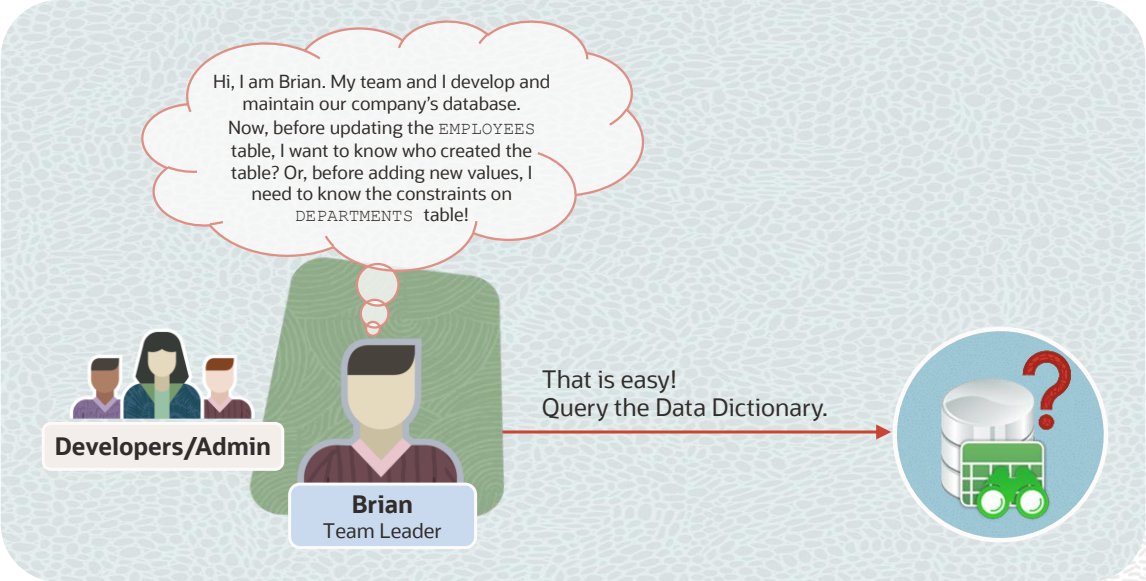
- Introduction to data dictionary
- Querying the dictionary views for the following:
  - Table information
  - Column information
  - Constraint information
- Adding a comment to a table and querying the dictionary views for comment information

4

This section gives a brief introduction to data dictionary.

Oracle Internal & Oracle Academy Use Only

Why Data Dictionary?



5

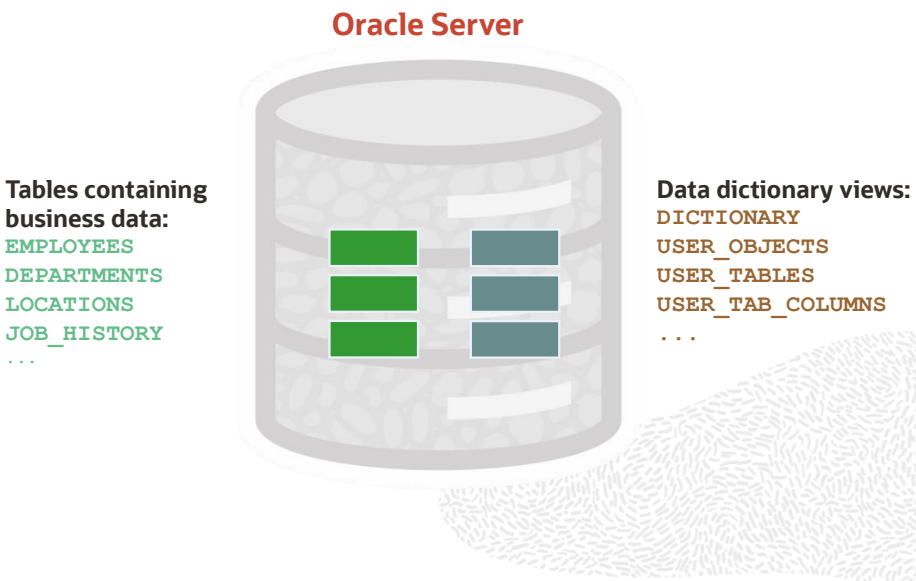
Brian and his team of developers are currently developing and maintaining his company's database. So, at some point in time, Brian (Team Leader) wants to alter the `EMPLOYEES` table by dropping/adding a column. Before altering the table, he wants to check with the owner of the table and discuss the effects of the change. He also wants to add a new department into the `DEPARTMENTS` table. Before inserting new values into the table, he needs to know the various constraints on the table so that they are not violated.

How do you think Brian will keep track of this information?

The good news is that Brian need not maintain any other document or table to store such information. All he needs to do is learn to query the data dictionary. The data dictionary is a list of in-built tables and views, which come along with the database. You learn more about data dictionary in the following slides.

Oracle Internal & Oracle Academy Use Only

Data Dictionary



6

User tables are tables created by you and contain business data, such as `EMPLOYEES`. Along with user tables, there is another collection of tables and views in the Oracle database known as the *data dictionary*. This collection is created and maintained by the Oracle Server and contains information about the database.

The data dictionary is structured in tables and views, just like other database data. The data dictionary is central to every Oracle database and is an important tool for all users, from end users to application designers and database administrators.

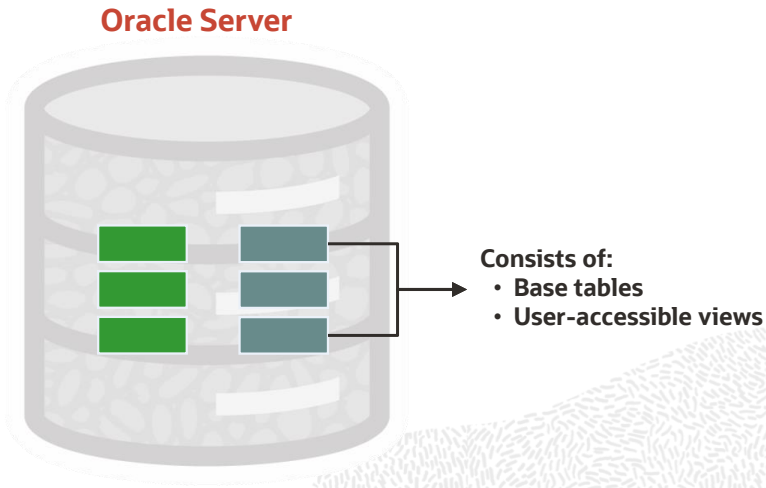
You use SQL statements to access the data dictionary. Remember that the data dictionary is read-only and, therefore, you can issue queries only against its tables and views.

You can query the dictionary views that are based on the dictionary tables to find information such as:

- Definitions of all schema objects in the database (tables, views, indexes, synonyms, sequences, procedures, functions, packages, triggers, and so on)
- Default values for columns
- Integrity constraint information
- Names of Oracle users
- Privileges and roles that each user has been granted
- Other general database information

Oracle Internal & Oracle Academy Use Only

Data Dictionary Structure



7

Underlying base tables store information about the associated database. Only the Oracle Server should write to and read from these tables. You rarely access them directly.

There are several views that summarize and display the information stored in the base tables of the data dictionary. These views decode the base table data into useful information (such as user or table names) using joins and `WHERE` clauses to simplify the information. You are mostly given access to the views rather than the base tables.

The Oracle user `SYS` owns all base tables and user-accessible views of the data dictionary. No Oracle user should *ever* alter (`UPDATE`, `DELETE`, or `INSERT`) any rows or schema objects contained in the `SYS` schema; doing so can compromise data integrity.

You will learn more about views and how to create them in the lesson titled “Creating Views.”

Oracle Internal & Oracle Academy Use Only

Data Dictionary Structure

View naming convention is as follows:

View Prefix	Purpose
USER	User's view (what is in your schema; what you own)
ALL	Expanded user's view (what you can access)
DBA	Database administrator's view (what is in everyone's schemas)
V$	Performance-related data

8

The data dictionary consists of sets of views. In many cases, a set consists of three views containing similar information and distinguished from each other by their prefixes. For example, there is a view named `USER_OBJECTS`, another named `ALL_OBJECTS`, and a third named `DBA_OBJECTS`.

These three views contain similar information about objects in the database, except that the scope is different.

`USER_OBJECTS` contains information about objects that you own or you created. `ALL_OBJECTS` contains information about all objects to which you have access. `DBA_OBJECTS` contains information about all objects that are owned by all users.

For views that are prefixed with `ALL` or `DBA`, you will find an additional column in the view named `OWNER` to identify who owns the object.

There is also a set of views that is prefixed with `v$`. These views are dynamic in nature and hold information about performance. Dynamic performance tables are not true tables, and they should not be accessed by most users. However, database administrators can query and create views on the tables and grant access to those views to other users. This course does not go into details about these views.

Oracle Internal & Oracle Academy Use Only

How to Use Dictionary Views

Start with DICTIONARY. It contains the names and descriptions of the dictionary tables and views.

```
DESCRIBE DICTIONARY
```

```
DESCRIBE dictionary
Name       Null Type
---------- ---- --------------
TABLE_NAME      VARCHAR2(128)
COMMENTS        VARCHAR2(4000)
```

```
SELECT *
FROM   dictionary
WHERE  table_name = 'USER_OBJECTS';
```

	TABLE_NAME	COMMENTS
1	USER_OBJECTS	Objects owned by the user

9

To familiarize yourself with the dictionary views, you can use the dictionary view named DICTIONARY. It contains the name and short description of each dictionary view to which you have access.

You can write queries to search for information about a particular view name, or you can search the COMMENTS column for a word or phrase. In the example shown in the slide, the DICTIONARY view is described. It has two columns. The SELECT statement in the slide retrieves information about the dictionary view named USER_OBJECTS. The USER_OBJECTS view contains information about all the objects that you own.

You can write queries to search the COMMENTS column for a word or phrase. For example, the following query returns the names of all views that you are permitted to access in which the COMMENTS column contains the word *columns*:

```
SELECT table_name
FROM dictionary
WHERE LOWER(comments) LIKE '%columns%';
```

Note: The table names in the data dictionary are in uppercase.

Oracle Internal & Oracle Academy Use Only

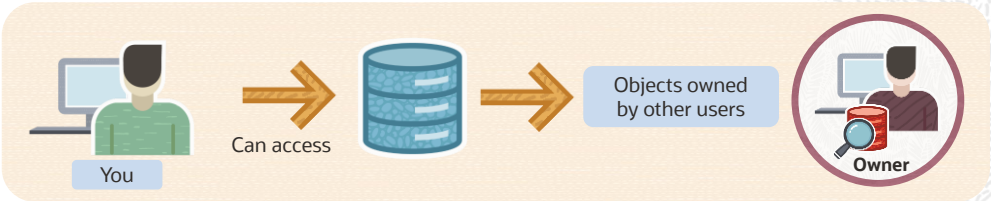
USER_OBJECTS and ALL_OBJECTS Views

USER_OBJECTS:

- Query USER_OBJECTS to see all the objects that you own.
- Using USER_OBJECTS, you can obtain a listing of all object names and types in your schema, plus the following information:
  - Date created
  - Date of last modification
  - Status (valid or invalid)

ALL_OBJECTS:

- Query ALL_OBJECTS to see all the objects to which you have access.



10

You can query the USER_OBJECTS view to see the names and types of all the objects in your schema. There are several columns in this view:

- **OBJECT_NAME:** Name of the object
- **OBJECT_ID:** Dictionary object number of the object
- **OBJECT_TYPE:** Type of object (such as TABLE, VIEW, INDEX, SEQUENCE)
- **CREATED:** Time stamp for the creation of the object
- **LAST_DDL_TIME:** Time stamp for the last modification of the object resulting from a data definition language (DDL) command
- **STATUS:** Status of the object (VALID, INVALID, or N/A)
- **GENERATED:** Was the name of this object system generated? (Y|N)

Note: This is not a complete listing of the columns. For a complete listing, see "USER_OBJECTS" in *Oracle® Database Reference 19c* .

You can also query the ALL_OBJECTS view to see a listing of all objects to which you have access.

Oracle Internal & Oracle Academy Use Only

USER_OBJECTS View

```
SELECT object_name, object_type, created, status
FROM   user_objects
ORDER BY object_type;
```

	OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
1	LOC_COUNTRY_IX	INDEX	27-JUN-16	VALID
2	EMP_DEPARTMENT_IX	INDEX	27-JUN-16	VALID
3	LOC_STATE_PROVINCE_IX	INDEX	27-JUN-16	VALID
4	COUNTRY_C_ID_PK	INDEX	27-JUN-16	VALID
5	LOC_CITY_IX	INDEX	27-JUN-16	VALID
6	LOC_ID_PK	INDEX	27-JUN-16	VALID
7	JHIST_DEPARTMENT_IX	INDEX	27-JUN-16	VALID
8	JHIST_EMPLOYEE_IX	INDEX	27-JUN-16	VALID
9	DEPT_ID_PK	INDEX	27-JUN-16	VALID

Owned by the user

...

11

The example in the slide shows the name, type, date of creation, and status of all objects that are owned by this user.

The `OBJECT_TYPE` column holds the values of `TABLE`, `VIEW`, `SEQUENCE`, `INDEX`, `PROCEDURE`, `FUNCTION`, `PACKAGE`, or `TRIGGER`.

The `STATUS` column holds a value of `VALID`, `INVALID`, or `N/A`. Although tables are always valid, the views, procedures, functions, packages, and triggers may be invalid.

The `CAT` View

For a simplified query and output, you can query the `CAT` view. This view contains only two columns:

- `TABLE_NAME`
- `TABLE_TYPE`

It provides the names of all your `INDEX`, `TABLE`, `CLUSTER`, `VIEW`, `SYNONYM`, `SEQUENCE`, or `UNDEFINED` objects.

Note: `CAT` is a synonym for `USER_CATALOG`—a view that lists tables, views, synonyms, and sequences owned by the user.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Introduction to data dictionary
- Querying the dictionary views for the following:
  - Table information
  - Column information
  - Constraint information
- Adding a comment to a table and querying the dictionary views for comment information

12

This section discusses about querying the dictionary views for table information, column information, and constraint information.

Oracle Internal & Oracle Academy Use Only

Table Information

USER_TABLES:

```
DESCRIBE user_tables
```

```
DESCRIBE user_tables
Name                    Null     Type
----------------------- -------- -------------
TABLE_NAME              NOT NULL VARCHAR2(128)
TABLESPACE_NAME                  VARCHAR2(30)
CLUSTER_NAME                     VARCHAR2(128)
IOT_NAME                         VARCHAR2(128)
...
```

```
SELECT table_name
FROM   user_tables;
```

	TABLE_NAME
1	REGIONS
2	LOCATIONS
3	DEPARTMENTS
4	JOBS
5	EMPLOYEES
6	JOB_HISTORY

...

13

You can use the USER_TABLES view to obtain the names of all your tables. The USER_TABLES view contains information about your tables. In addition to providing the table name, it contains detailed information about the storage.

The TABS view is a synonym of the USER_TABLES view. You can query it to see a listing of tables that you own:

```
SELECT table_name
FROM  tabs;
```

Note: For a complete listing of the columns in the USER_TABLES view, see "USER_TABLES" in *Oracle® Database Reference 19c.*

You can also query the ALL_TABLES view to see a listing of all tables to which you have access.

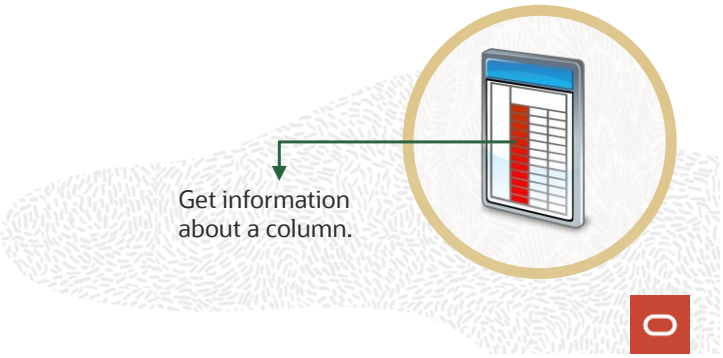
Oracle Internal & Oracle Academy Use Only

Column Information

USER_TAB_COLUMNS:

```
DESCRIBE user_tab_columns
```

```
Name                   Null     Type
---------------------- -------- --------------
TABLE_NAME             NOT NULL VARCHAR2(128)
COLUMN_NAME            NOT NULL VARCHAR2(128)
DATA_TYPE                       VARCHAR2(128)
DATA_TYPE_MOD                   VARCHAR2(3)
DATA_TYPE_OWNER                 VARCHAR2(128)
DATA_LENGTH            NOT NULL NUMBER
DATA_PRECISION                  NUMBER
DATA_SCALE                      NUMBER
NULLABLE                        VARCHAR2(1)
...
```



14

You can query the USER_TAB_COLUMNS view to find detailed information about the columns in your tables. Although the USER_TABLES view provides information about your table names and storage, you will find detailed column information in the USER_TAB_COLUMNS view.

This view contains information such as:

- Column names
- Column data types
- Length of data types
- Precision and scale for NUMBER columns
- Whether nulls are allowed (Is there a NOT NULL constraint on the column?)
- Default value

Note: For a complete listing and description of the columns in the USER_TAB_COLUMNS view, see "USER_TAB_COLUMNS" in the *Oracle® Database Reference 19c.*

Oracle Internal & Oracle Academy Use Only

Column Information

```
SELECT column_name, data_type, data_length,
       data_precision, data_scale, nullable
FROM   user_tab_columns
WHERE  table_name = 'EMPLOYEES';
```

	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NULLABLE
1	EMPLOYEE_ID	NUMBER	22	6	0	N
2	FIRST_NAME	VARCHAR2	20	(null)	(null)	Y
3	LAST_NAME	VARCHAR2	25	(null)	(null)	N
4	EMAIL	VARCHAR2	25	(null)	(null)	N
5	PHONE_NUMBER	VARCHAR2	20	(null)	(null)	Y
6	HIRE_DATE	DATE	7	(null)	(null)	N
7	JOB_ID	VARCHAR2	10	(null)	(null)	N
8	SALARY	NUMBER	22	8	2	Y
9	COMMISSION_PCT	NUMBER	22	2	2	Y
10	MANAGER_ID	NUMBER	22	6	0	Y
11	DEPARTMENT_ID	NUMBER	22	4	0	Y

15

By querying the `USER_TAB_COLUMNS` table, you can find details about your columns, such as the names, data types, data type lengths, null constraints, and default value for a column.

The example shown in the slide displays the columns, data types, data lengths, and null constraints for the `EMPLOYEES` table. Note that this information is similar to the output from the `DESCRIBE` command.

To view information about columns set as unused, you use the `USER_UNUSED_COL_TABS` dictionary view.

Note: Names of the objects in data dictionary are in uppercase.

Oracle Internal & Oracle Academy Use Only

Constraint Information

- `USER_CONSTRAINTS` describes the constraint definitions on your tables.
- `USER_CONS_COLUMNS` describes columns that are owned by you and that are specified in constraints.

```
DESCRIBE user_constraints
```

```
Name                 Null     Type
-------------------- -------- --------------
OWNER                         VARCHAR2(128)
CONSTRAINT_NAME      NOT NULL VARCHAR2(128)
CONSTRAINT_TYPE               VARCHAR2(1)
TABLE_NAME           NOT NULL VARCHAR2(128)
SEARCH_CONDITION              LONG
SEARCH_CONDITION_VC           VARCHAR2(4000)
R_OWNER                       VARCHAR2(128)
R_CONSTRAINT_NAME             VARCHAR2(128)
DELETE_RULE                   VARCHAR2(9)
STATUS                        VARCHAR2(8)
...
```

16

You can find out the names of your constraints, the type of constraint, the table name to which the constraint applies, the condition for check constraints, foreign key constraint information, deletion rule for foreign key constraints, the status, and many other types of information about your constraints.

Note: For a complete listing and description of the columns in the `USER_CONSTRAINTS` view, see "`USER_CONSTRAINTS`" in *Oracle® Database Reference 19c.*

Oracle Internal & Oracle Academy Use Only

USER_CONSTRAINTS: Example

```
SELECT constraint_name, constraint_type,
       search_condition, r_constraint_name,
       delete_rule, status
FROM   user_constraints
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
1	EMP_MANAGER_FK	R	(null)	EMP_EMP_ID_PK	NO ACTION	ENABLED
2	EMP_JOB_FK	R	(null)	JOB_ID_PK	NO ACTION	ENABLED
3	EMP_DEPT_FK	R	(null)	DEPT_ID_PK	NO ACTION	ENABLED
4	EMP_EMP_ID_PK	P	(null)	(null)	(null)	ENABLED
5	EMP_EMAIL_UK	U	(null)	(null)	(null)	ENABLED
6	EMP_SALARY_MIN	C	salary > 0	(null)	(null)	ENABLED
7	EMP_JOB_NN	C	"JOB_ID" IS NOT NULL	(null)	(null)	ENABLED
8	EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL	(null)	(null)	ENABLED
9	EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL	(null)	(null)	ENABLED
10	EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL	(null)	(null)	ENABLED

17

In the example in the slide, the USER_CONSTRAINTS view is queried to find the names, types, check conditions, name of the unique constraint that the foreign key references, deletion rule for a foreign key, and status for constraints on the EMPLOYEES table.

The CONSTRAINT_TYPE can be:

- C (check constraint on a table, or NOT NULL)
- P (primary key)
- U (unique key)
- R (referential integrity)
- V (with check option, on a view)
- O (with read-only, on a view)

The DELETE_RULE can be:

- **CASCADE:** If the parent record is deleted, the child records are deleted too.
- **SET NULL:** If the parent record is deleted, change the respective child record to null.
- **NO ACTION:** A parent record can be deleted only if no child records exist.

The STATUS can be:

- **ENABLED:** Constraint is active.
- **DISABLED:** Constraint is made inactive.

Oracle Internal & Oracle Academy Use Only

Querying USER_CONS_COLUMNS

```
DESCRIBE user_cons_columns
```

```
DESCRIBE user_cons_columns
Name            Null     Type
--------------- -------- --------------
OWNER           NOT NULL VARCHAR2(128)
CONSTRAINT_NAME NOT NULL VARCHAR2(128)
TABLE_NAME      NOT NULL VARCHAR2(128)
COLUMN_NAME              VARCHAR2(4000)
POSITION                 NUMBER
```

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	COLUMN_NAME
1	EMP_LAST_NAME_NN	LAST_NAME
2	EMP_EMAIL_NN	EMAIL
3	EMP_HIRE_DATE_NN	HIRE_DATE
4	EMP_JOB_NN	JOB_ID
5	EMP_SALARY_MIN	SALARY
6	EMP_EMAIL_UK	EMAIL
7	EMP_EMP_ID_PK	EMPLOYEE_ID
8	EMP_DEPT_FK	DEPARTMENT_ID
9	EMP_JOB_FK	JOB_ID
10	EMP_MANAGER_FK	MANAGER_ID

...

18

To find the names of the columns to which a constraint applies, query the `USER_CONS_COLUMNS` dictionary view. This view tells you the name of the owner of a constraint, the name of the constraint, the table that the constraint is on, the names of the columns with the constraint, and the original position of column or attribute in the definition of the object.

Note: A constraint may apply to more than one column.

You can also write a join between `USER_CONSTRAINTS` and `USER_CONS_COLUMNS` to create customized output from both tables.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Introduction to data dictionary
- Querying the dictionary views for the following:
  - Table information
  - Column information
  - Constraint information
- Adding a comment to a table and querying the dictionary views for comment information

19

In this section, you learn how to add a comment to a table and how to view the comment by using dictionary views.

Oracle Internal & Oracle Academy Use Only

Adding Comments to a Table

- You can add comments to a table or column by using the COMMENT statement:

```
COMMENT ON TABLE employees
IS 'Employee Information';
```

```
COMMENT ON COLUMN employees.first_name
IS 'First name of the employee';
```

- Comments can be viewed through the data dictionary views:
  - ALL_COL_COMMENTS
  - USER_COL_COMMENTS
  - ALL_TAB_COMMENTS
  - USER_TAB_COMMENTS

20

You can add a comment of up to 4,000 bytes about a column, table, view, or snapshot by using the COMMENT statement. The comment is stored in the data dictionary and can be viewed in one of the following data dictionary views in the COMMENTS column:

- ALL_COL_COMMENTS
- USER_COL_COMMENTS
- ALL_TAB_COMMENTS
- USER_TAB_COMMENTS

Syntax

```
COMMENT ON {TABLE table | COLUMN table.column}
    IS 'text';
```

In the syntax:

table	Is the name of the table
column	Is the name of the column in a table
text	Is the text of the comment

You can drop a comment from the database by setting it to empty string (''):

```
COMMENT ON TABLE  employees IS '';
```

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to find information about your objects by using the following dictionary views:

- `DICTIONARY`
- `USER_OBJECTS`
- `USER_TABLES`
- `USER_TAB_COLUMNS`
- `USER_CONSTRAINTS`
- `USER_CONS_COLUMNS`

21

In this lesson, you learned about some of the dictionary views that are available to you. You can use these dictionary views to find information about your tables, constraints, views, sequences, and synonyms.

Oracle Internal & Oracle Academy Use Only

Practice 12: Overview

This practice covers the following topics:

- Querying the dictionary views for table and column information
- Querying the dictionary views for constraint information
- Adding a comment to a table and querying the dictionary views for comment information

22

In this practice, you query the dictionary views to find information about objects in your schema.

Oracle Internal & Oracle Academy Use Only

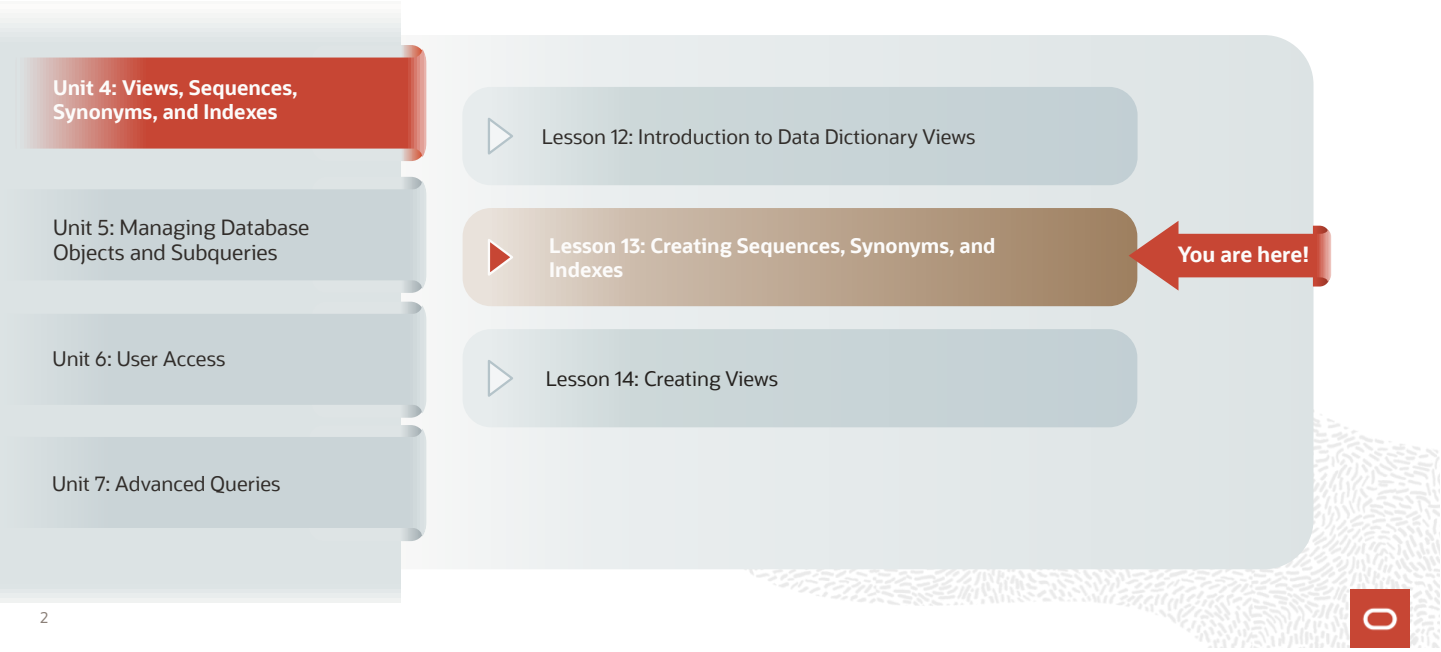
ORACLE

13

Creating Sequences, Synonyms, and Indexes

Oracle Internal & Oracle Academy Use Only

Course Roadmap



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, indexes, and how to create them.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Create, maintain, and use sequences
- Create private and public synonyms
- Create and maintain indexes
- Query various data dictionary views to find information for sequences, synonyms, and indexes

3

In this lesson, you are introduced to the sequence, synonyms, and index objects. You learn the basics of creating and using sequences, synonyms, and indexes.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
  - SQL column defaulting using a sequence
- Overview of synonyms
  - Creating and dropping synonyms
- Overview of indexes
  - Creating indexes
  - Using the CREATE TABLE statement
  - Creating function-based indexes
  - Creating multiple indexes on the same set of columns
  - Removing indexes

4

This section discusses the overview of sequences. You will learn how to create, modify, and use sequences. You will also learn about the `NEXTVAL` and `CURRVAL` pseudocolumns and how to use them in creating default values for a column.

Oracle Internal & Oracle Academy Use Only

E-Commerce Scenario

OracleKart is an online e-commerce company selling a variety of goods. The back-end database maintains a table for all orders placed by the customers. Each order requires a unique ID to store the order information correctly. How do you think OracleKart will generate the `order_id` in the `ORDERS` table? Should an administrator manually insert a unique `order_id` each time an order is placed?

Brian, the DBA, is aware that manually generating unique IDs is not efficient. An error can cause incorrect data mapping, which will lead to incorrect order delivery. For data integrity and work efficiency, Brian can make use of SQL sequence to generate order IDs automatically.

To solve scenarios like the above, SQL provides sequences. Using a sequence, you can generate a unique sequence of numbers according to your need. Let us look into some more advantages of a sequence and learn how to use it in a SQL query.

Oracle Internal & Oracle Academy Use Only

Database Objects

Object	Description
Table	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 data retrieval queries
Synonym	Gives alternative names to objects

6

As shown in the slide, there are several other objects in a database in addition to **tables**.

With **views**, you can present and hide data from the tables.

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a **sequence** to generate unique numbers.

If you want to improve the performance of data retrieval queries, you should consider creating an **index**. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using **synonyms**.

Oracle Internal & Oracle Academy Use Only

Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.

```
SELECT *
FROM userB.employees;
```

```
SELECT *
FROM userA.employees;
```

7

What is a schema?

A schema is a collection of logical structures of data or *schema objects*. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.

Schema objects can be created and manipulated with SQL and include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.

If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there are schemas named `USERA` and `USERB`, and both have an `EMPLOYEES` table, then if `USERA` wants to access the `EMPLOYEES` table that belongs to `USERB`, `USERA` must prefix the table name with the schema name:

```
SELECT *
FROM   userb.employees;
```

If `USERB` wants to access the `EMPLOYEES` table that is owned by `USERA`, `USERB` must prefix the table name with the schema name:

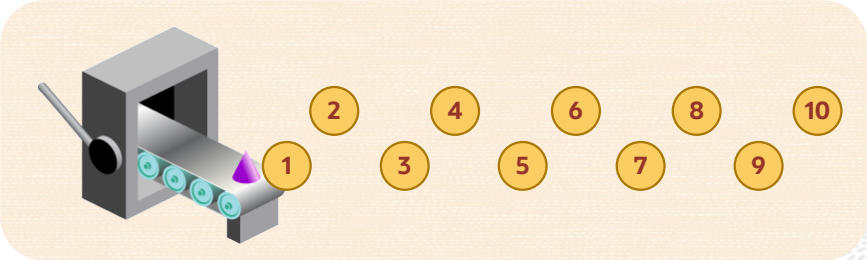
```
SELECT *
FROM   usera.employees;
```

Oracle Internal & Oracle Academy Use Only

Sequences

A sequence:

- Can automatically generate unique numbers
- Is a shareable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory



8

A sequence is a user-created database object that can be shared by multiple users to generate integers.

You can define a sequence to generate unique values or to recycle and use the same numbers again.

A typical usage for sequences is to create a primary key value, which must be unique for each row. A sequence is generated and incremented (or decremented) by an internal Oracle routine. This can be a time-saving object, because it can reduce the amount of application code needed to write a sequence-generating routine.

Sequence numbers are stored and generated independent of tables. Therefore, the same sequence can be used for multiple tables.

Oracle Internal & Oracle Academy Use Only

CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE [ schema. ] sequence
  [ { START WITH|INCREMENT BY } integer
  | { MAXVALUE integer | NOMAXVALUE }
  | { MINVALUE integer | NOMINVALUE }
  | { CYCLE | NOCYCLE }
  | { CACHE integer | NOCACHE }
  | { ORDER | NOORDER }
  | { SCALE | NOSCALE }
  | { SHARD | NOSHARD }
];
```

9

Automatically generate sequential numbers by using the `CREATE SEQUENCE` statement.

In the syntax:

Sequence	Is the name of the sequence generator
`START WITH` *n*	Specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)
`INCREMENT BY` *n*	Specifies the interval between sequence numbers, where *n* is an integer (If this clause is omitted, the sequence increments by 1.)
`MAXVALUE` *n*	Specifies the maximum value the sequence can generate
`NOMAXVALUE`	Specifies a maximum value of 10^27 for an ascending sequence and –1 for a descending sequence (This is the default option.)
`MINVALUE` *n*	Specifies the minimum sequence value.
`NOMINVALUE`	Specifies a minimum value of 1 for an ascending sequence and –(10^26) for a descending sequence (This is the default option.)

Oracle Internal & Oracle Academy Use Only

ORDER If specified, guarantees that sequence numbers are generated in order of request. This clause is useful if you are using the sequence numbers as timestamps.

NOORDER If specified, sequence numbers are not generated in order of request. This is the default.

CYCLE | NOCYCLE Specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)

CACHE *n* | NOCACHE Specifies how many values the Oracle Server pre-allocates and keeps in memory (By default, the Oracle server caches 20 values.)

SCALE [EXTEND|NOEXTEND] | NOSCALE If specified, a numeric offset is affixed to the beginning of the sequence, which removes all duplicates in generated values (NOSCALE is the default option.)

If EXTEND is specified with SCALE, the generated sequence values are all of length (x+y), where x is the length of the scalable offset (default value is 6) and y is the maximum number of digits in the sequence (maxvalue/minvalue) (NOEXTEND is the default).

SHARD [EXTEND | NOEXTEND] | NOSHARD If specified, the sequence generates unique sequence numbers across shards. If EXTEND is specified with the SHARD clause, the generated sequence values are all of length (x = y), where *x* is the length of a SHARD offset of size 4. (NOEXTEND is the default.)

Oracle Internal & Oracle Academy Use Only

Creating a Sequence

- Create a sequence named `DEPT_DEPTID_SEQ` to be used for the primary key of the `DEPARTMENTS` table.
- Do not use the `CYCLE` option.

```
CREATE SEQUENCE dept_deptid_seq
                START WITH 280
                INCREMENT BY 10
                MAXVALUE 9999
                NOCACHE
                NOCYCLE
                SCALE;
```

```
Sequence DEPT_DEPTID_SEQ created.
```

11

The example in the slide creates a sequence named DEPT_DEPTID_SEQ to be used for the DEPARTMENT_ID column of the DEPARTMENTS table. The sequence starts at 280 and adds a 6-digit prefix to the sequence number (three-digit instance number followed by a three-digit session number). It does not allow caching and does not cycle.

Do not use the CYCLE option if the sequence is used to generate primary key values, unless you have a reliable mechanism that purges old rows faster than the sequence cycles.

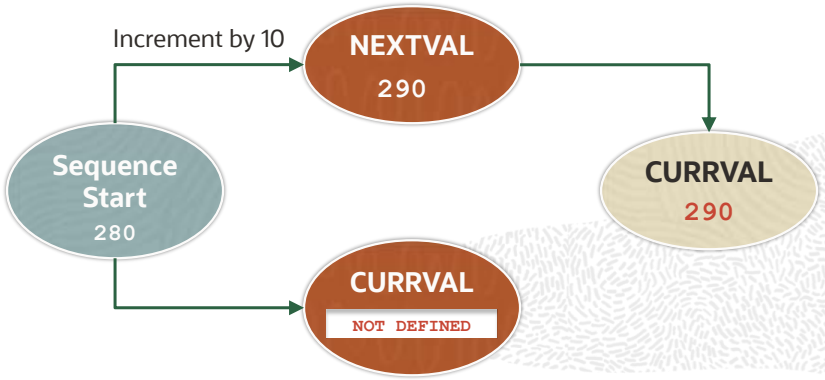
For more information, see the "`CREATE SEQUENCE`" section in *Oracle Database SQL Language Reference* for Oracle Database 19c.

Note: The sequence is not tied to a table. Generally, you should name the sequence after its intended use. However, the sequence can be used anywhere, regardless of its name.

Oracle Internal & Oracle Academy Use Only

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL can be referenced.



12

After you create your sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference *sequence*.NEXTVAL, a new sequence number is generated, and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. However, NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced. You must qualify CURRVAL with the sequence name. When you reference *sequence*.CURRVAL, the last value returned to that user's process is displayed.

Rules for Using NEXTVAL and CURRVAL

You can use NEXTVAL and CURRVAL in the following contexts:

- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

Oracle Internal & Oracle Academy Use Only

You cannot use `NEXTVAL` and `CURRVAL` in the following contexts:

- The `SELECT` list of a view
- A `SELECT` statement with the `DISTINCT` keyword
- A `SELECT` statement with `GROUP BY`, `HAVING`, or `ORDER BY` clause
- A subquery in a `SELECT`, `DELETE`, or `UPDATE` statement

For more information, see the "Pseudocolumns" and "`CREATE SEQUENCE`" sections in *Oracle Database SQL Language Reference* for Oracle Database 19*c.*

Oracle Internal & Oracle Academy Use Only

Using a Sequence

- Insert a new department named “Support” in location ID 2500:

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
```

```
1 row inserted.
```

- View the current value for the `DEPT_DEPTID_SEQ` sequence:

```
SELECT dept_deptid_seq.CURRVAL
FROMdual;
```

	CURRVAL
1	280

14

The example in the slide inserts a new department in the `DEPARTMENTS` table. It uses the `DEPT_DEPTID_SEQ` sequence to generate a new department number as shown in the slide.

You can view the current value of the sequence using the *sequence_name*.`CURRVAL`, as shown in the second example in the slide.

Suppose that you now want to hire employees to staff the new department. The `INSERT` statement to be executed for all new employees can include the following code:

```
INSERT INTO employees (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq .CURRVAL, ...);
```

Note: The preceding example assumes that a sequence called `EMPLOYEES_SEQ` has already been created to generate new employee numbers.

Oracle Internal & Oracle Academy Use Only

SQL Column Defaulting Using a Sequence

- You can use the SQL syntax `<sequence>.nextval, <sequence>.currval` as a SQL column defaulting expression for numeric columns, where <sequence> is an Oracle database sequence.
- The `DEFAULT` expression can include the sequence pseudocolumns `CURRVAL` and `NEXTVAL`, as long as the sequence exists and you have the privileges necessary to access it.

```
CREATE SEQUENCE ID_SEQ START WITH 1;
CREATE TABLE emp (ID NUMBER DEFAULT ID_SEQ.NEXTVAL NOT NULL,
                  name VARCHAR2(10));
INSERT INTO emp (name) VALUES ('john');
INSERT INTO emp (name) VALUES ('mark');
SELECT * FROM emp;
```

```
Sequence ID_SEQ created.
Table EMP created.

1 row inserted.
1 row inserted.

        ID NAME
---------- ----------
         1 john
         2 mark
```

15

SQL syntax for column defaults has been enhanced so that it allows `<sequence>.nextval, <sequence>.currval` as a SQL column defaulting expression for numeric columns, where `<sequence>` is an Oracle database sequence.

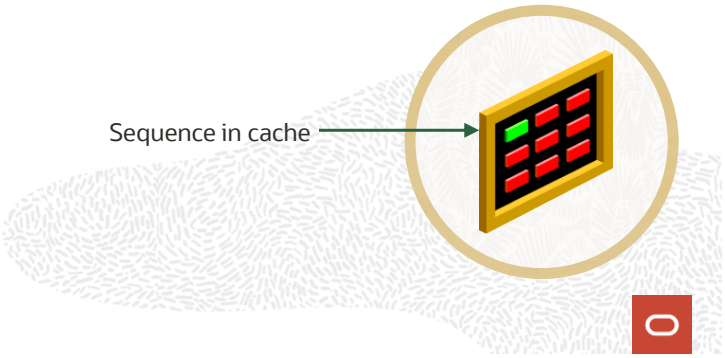
The `DEFAULT` expression can include the sequence pseudocolumns `CURRVAL` and `NEXTVAL`, as long as the sequence exists and you have the privileges necessary to access it. The user inserting into a table must have access privileges to the sequence. If the sequence is dropped, subsequent insert DMLs where *expr* is used for defaulting will result in a compilation error.

In the example in the slide, sequence `ID_SEQ` is created, which starts from 1.

Oracle Internal & Oracle Academy Use Only

Caching Sequence Values

- Caching sequence values in memory gives faster access to those values.
- Gaps in sequence values can occur when:
  - A rollback occurs
  - The system crashes
  - A sequence is used in another table



16

You can cache sequences in memory to provide faster access to those sequence values. The cache is populated the first time you refer to the sequence. Each request for the next sequence value is retrieved from the cached sequence. After the last sequence value is used, the next request for the sequence pulls another cache of sequences into memory.

Gaps in the Sequence

The sequence generators generally issue sequential numbers without gaps, and this action occurs independently of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.

Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in memory, those values are lost if the system crashes.

Because sequences are not tied directly to tables, the same sequence can be used for multiple tables. However, if you do so, each table can contain gaps in the sequential numbers.

Oracle Internal & Oracle Academy Use Only

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE
               NOSCALE;
```

```
Sequence DEPT_DEPTID_SEQ altered.
```

17

If you reach the `MAXVALUE` limit for your sequence, no additional values from the sequence are allocated and you will receive an error indicating that the sequence exceeds the `MAXVALUE`. To continue to use the sequence, you can modify it by using the `ALTER SEQUENCE` statement.

Syntax

```
ALTER  SEQUENCE  sequence
      [INCREMENT BY n]
      [{MAXVALUE n | NOMAXVALUE}]
      [{MINVALUE n | NOMINVALUE}]
      [{CYCLE | NOCYCLE}]
      [{CACHE n | NOCACHE}]
      [{SCALE [EXTEND | NOEXTEND] | NOSCALE}]
      [{SHARD [EXTEND | NOEXTEND] | NOSHARD;
```

In the syntax, *`sequence`* is the name of the sequence generator.

For more information, see the section on "`ALTER SEQUENCE`" in *Oracle Database SQL Language Reference* for Oracle Database 19*c*.

Oracle Internal & Oracle Academy Use Only

Guidelines for Modifying a Sequence

- You must be the owner or have the ALTER privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the DROP statement:

```
DROP SEQUENCE dept_deptid_seq;
```

```
Sequence DEPT_DEPTID_SEQ dropped.
```

18

- You must be the owner or have the ALTER privilege for the sequence to modify it. You must be the owner or have the DROP ANY SEQUENCE privilege to remove it.
- Only future sequence numbers are affected by the ALTER SEQUENCE statement.
- The START WITH option cannot be changed by using ALTER SEQUENCE. The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed. For example, a new MAXVALUE that is less than the current sequence number cannot be imposed.

```
ALTER SEQUENCE dept_deptid_seq
      INCREMENT BY 20
      MAXVALUE 90
      NOCACHE
      NOCYCLE
      NOSCALE;
```

```
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current
value
04009. 00000 - "MAXVALUE cannot be made to be less than the current value"
*Cause:   the current value exceeds the given MAXVALUE
*Action:  make sure that the new MAXVALUE is larger than the current value
```

Oracle Internal & Oracle Academy Use Only

Sequence Information

- The USER_SEQUENCES view describes all sequences that you own.

```
DESCRIBE user_sequences
```

```
SEQUENCE_NAME   VARCHAR2(128)
MIN_VALUE       NUMBER
MAX_VALUE       NUMBER
INCREMENT_BY    NUMBER
CYCLE_FLAG      VARCHAR2(1)
ORDER_FLAG      VARCHAR2(1)
CACHE_SIZE      NUMBER
LAST_NUMBER     NUMBER
SCALE_FLAG      VARCHAR2(1)
EXTEND_FLAG     VARCHAR2(1)
SHARDED_FLAG    VARCHAR2(1)
SESSION_FLAG    VARCHAR2(1)
KEEP_VALUE      VARCHAR2(1)
```

- Verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT sequence_name, min_value, max_value,
increment_by, last_number
FROM user_sequences;
```

19

The USER_SEQUENCES view describes all sequences that you own. When you create the sequence, you specify criteria that are stored in the USER_SEQUENCES view. The following are the columns in this view:

- SEQUENCE_NAME: Name of the sequence
- MIN_VALUE: Minimum value of the sequence
- MAX_VALUE: Maximum value of the sequence
- INCREMENT_BY: Value by which the sequence is incremented
- CYCLE_FLAG: Whether sequence wraps around on reaching the limit
- ORDER_FLAG: Whether sequence numbers are generated in order
- CACHE_SIZE: Number of sequence numbers to cache
- LAST_NUMBER: Last sequence number written to disk. If a sequence uses caching, the number written to disk is the last number placed in the sequence cache. If NOCACHE is specified, the LAST_NUMBER column displays the next available sequence number
- SCALE_FLAG: Indicates whether this is a scalable sequence (Y) or not (N)
- EXTEND_FLAG: Indicates whether this scalable sequence's generated values extend beyond MAX_VALUE or MIN_VALUE (Y) or not (N)
- SHARDED_FLAG: Indicates whether this is a sharded sequence (Y) or not (N)
- SESSION_FLAG: Whether the order value is session private
- KEEP_VALUE: Response time after an error

After creating your sequence, it is documented in the data dictionary. Because a sequence is a database object, you can identify it in the USER_OBJECTS data dictionary table.

You can also confirm the settings of the sequence by selecting from the USER_SEQUENCES data dictionary view.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
  - SQL column defaulting using a sequence
- Overview of synonyms
  - Creating and dropping synonyms
- Overview of indexes
  - Creating indexes
  - Using the CREATE TABLE statement
  - Creating function-based indexes
  - Creating multiple indexes on the same set of columns
  - Removing indexes

20

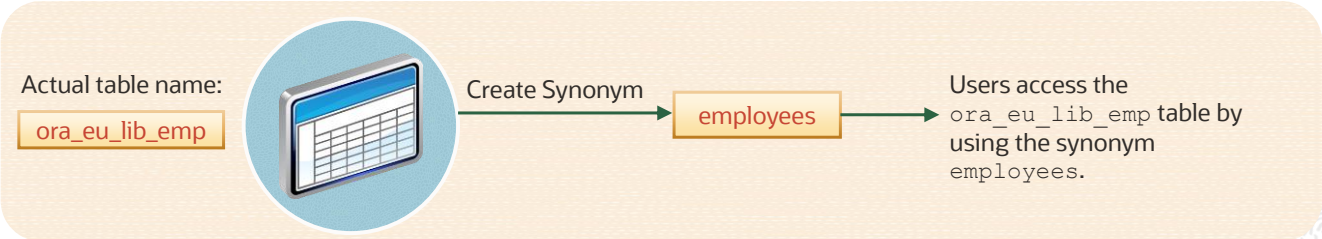
This section provides an overview of synonyms. You also learn how to create and drop synonyms.

Oracle Internal & Oracle Academy Use Only

Synonyms

A synonym:

- Is a database object
- Can be created to give an alternative name to a table or to another database object
- Requires no storage other than its definition in the data dictionary
- Is useful for hiding the identity and location of an underlying schema object



21

Synonyms are database objects that enable you to call a table by another name.

You can create synonyms to give an alternative name to a table or to another database object. For example, you can create a synonym for a table or view, sequence, PL/SQL program unit, user-defined object type, or another synonym.

Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms can simplify SQL statements for database users. Synonyms are also useful for hiding the identity and location of an underlying schema object.

Oracle Internal & Oracle Academy Use Only

Creating a Synonym for an Object

- You can simplify access to objects by creating a synonym (another name for an object).
- With synonyms, you can:
  - Create an easier reference to a table that is owned by another user
  - Shorten lengthy object names

```
CREATE [PUBLIC] SYNONYM synonym
FOR    object;
```

22

To refer to a table that is owned by another user, you need to prefix the table name with the name of the user who created it, followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:

`PUBLIC`	Creates a synonym that is accessible to all users
`synonym`	Is the name of the synonym to be created
`object`	Identifies the object for which the synonym is created

Guidelines

- The object cannot be contained in a package.
- A private synonym name must be distinct from all other objects that are owned by the same user.
- To create a `PUBLIC` synonym, you must have the `CREATE PUBLIC SYNONYM` system privilege.

For more information, see the section on "`CREATE SYNONYM`" in *Oracle Database SQL Language Reference* for Oracle Database 19*c*.

Oracle Internal & Oracle Academy Use Only

Creating and Removing Synonyms

- Create a shortened name for the DEPARTMENTS table:

```
CREATE SYNONYM  dept
FOR  departments;
```

```
Synonym DEPT created.
```

- Drop a synonym:

```
DROP SYNONYM dept;
```

```
Synonym DEPT dropped.
```

23

Oracle Internal & Oracle Academy Use Only

Creating a Synonym

The example in the slide creates a synonym for the DEPARTMENTS table for quicker reference.

The database administrator can create a public synonym that is accessible to all users. The following example creates a public synonym named DEPT for Alice's DEPARTMENTS table:

```
CREATE PUBLIC SYNONYM  dept
FOR    alice.departments;
```

Removing a Synonym

To remove a synonym, use the DROP SYNONYM statement. Only the database administrator can drop a public synonym.

```
DROP PUBLIC SYNONYM  dept;
```

For more information, see the section on "DROP SYNONYM" in *Oracle Database SQL Language Reference* for Oracle Database 19*c*.

Synonym Information

```
DESCRIBE user_synonyms
```

```
Name          Null     Type
------------- -------- -------------
SYNONYM_NAME  NOT NULL VARCHAR2(128)
TABLE_OWNER            VARCHAR2(128)
TABLE_NAME    NOT NULL VARCHAR2(128)
DB_LINK                VARCHAR2(128)
ORIGIN_CON_ID          NUMBER
```

```
SELECT *
FROM   user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK	ORIGIN_CON_ID
1	DEPT	TEACH_B	DEPARTMENTS	(null)	3

24

The `USER_SYNONYMS` dictionary view describes private synonyms (synonyms that you own).

You can query this view to find your synonyms. You can query `ALL_SYNONYMS` to find out the name of all the synonyms that are available to you and the objects on which these synonyms apply.

The columns in this view are:

- `SYNONYM_NAME`: Name of the synonym
- `TABLE_OWNER`: Owner of the object that is referenced by the synonym
- `TABLE_NAME`: Name of the table or view that is referenced by the synonym
- `DB_LINK`: Name of the database link reference (if any)
- `ORIGIN_CON_ID`: The container ID of the PDB where the data originates. In this case, it is 3.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
  - SQL column defaulting using a sequence
- Overview of synonyms
  - Creating and dropping synonyms
- Overview of indexes
  - Creating indexes
  - Using the CREATE TABLE statement
  - Creating function-based indexes
  - Creating multiple indexes on the same set of columns
  - Removing indexes

25

This section gives an overview of indexes. We will discuss how to create different types of indexes and how to remove indexes.

Oracle Internal & Oracle Academy Use Only

Indexes

An index:

- Is a schema object
- Can be used by the Oracle Server to speed up the retrieval of rows by using a pointer
- Can reduce disk input/output (I/O) by using a rapid path access method to locate data quickly
- Is dependent on the table that it indexes
- Is used and maintained automatically by the Oracle Server

26

You can use an index to speed up the retrieval of rows by using a pointer and improve the performance of some queries.

Indexes can be created explicitly or automatically. If you do not have an index on the column, a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the disk I/O by using an indexed path to locate data quickly. An index is used and maintained automatically by the Oracle Server. After an index is created, no direct activity is required by the user.

Indexes are logically and physically independent of the data in the objects with which they are associated. This means that they can be created or deleted at any time and have no effect on the base tables or other indexes.

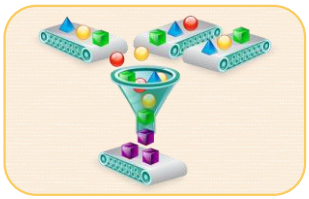
Note: When you drop a table, the corresponding indexes are also dropped.

For more information, see the section on “Schema Objects: Indexes” in *Oracle Database Concepts* 19c.

Oracle Internal & Oracle Academy Use Only

How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.



- Manually: You can create a unique or nonunique index on columns to speed up access to the rows.

27

You can create two types of indexes.

- **Unique index:** The Oracle Server automatically creates this index when you define a column in a table to have a `PRIMARY KEY` or a `UNIQUE` constraint. The name of the index is the name that is given to the constraint.
- **Nonunique index:** This is an index that a user can create. For example, you can create the `FOREIGN KEY` column index for a join in a query to improve the speed of retrieval.

Note: You can manually create a unique index, but it is recommended that you create a unique constraint, which implicitly creates a unique index.

Oracle Internal & Oracle Academy Use Only

Creating an Index

- Create an index on one or more columns:

```
CREATE [UNIQUE]INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the `LAST_NAME` column in the `EMPLOYEES` table:

```
CREATE INDEX   emp_last_name_idx
ON employees(last_name);
```

```
Index EMP_LAST_NAME_IDX created.
```

28

Create an index on one or more columns by issuing the `CREATE INDEX` statement.

In the syntax:

- `Index` Is the name of the index
- `Table` Is the name of the table
- `Column` Is the name of the column in the table to be indexed

Specify `UNIQUE` to indicate that the value of the column (or columns) on which the index is based must be unique. Specify `BITMAP` to indicate that the index is to be created with a bitmap for each distinct key, rather than indexing each row separately. Bitmap indexes store the `rowid` associated with a key value as a bitmap.

For more information, see the section on "`CREATE INDEX`" in *Oracle Database SQL Language Reference* for Oracle Database 19*c*.

Oracle Internal & Oracle Academy Use Only

CREATE INDEX with the CREATE TABLE Statement

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
             PRIMARY KEY USING INDEX
             (CREATE INDEX emp_id_idx ON
             NEW_EMP(employee_id)),
first_name   VARCHAR2(20),
last_name    VARCHAR2(25));
```

```
Table NEW_EMP created.
```

```
SELECT INDEX_NAME, TABLE_NAME
FROM   USER_INDEXES
WHERE  TABLE_NAME = 'NEW_EMP';
```

	INDEX_NAME	TABLE_NAME
1	EMP_ID_IDX	NEW_EMP

29

In the example in the slide, the CREATE INDEX clause is used with the CREATE TABLE statement to create a PRIMARY KEY index explicitly. You can name your indexes at the time of PRIMARY KEY creation to be different from the name of the PRIMARY KEY constraint.

You can query the USER_INDEXES data dictionary view for information about your indexes.

The following example illustrates the database behavior if the index is not explicitly named:

```
CREATE TABLE EMP_UNNAMED_INDEX
   (employee_id NUMBER(6) PRIMARY KEY ,
    first_name VARCHAR2(20),
    last_name VARCHAR2(25));
```

```
SELECT INDEX_NAME, TABLE_NAME
FROM   USER_INDEXES
WHERE  TABLE_NAME = 'EMP_UNNAMED_INDEX';
```

Oracle Internal & Oracle Academy Use Only

Observe that the Oracle Server gives a generic name to the index that is created for the `PRIMARY KEY` column.

You can also use an existing index for your `PRIMARY KEY` column—for example, when you are expecting a large data load and want to speed up the operation. You may want to disable the constraints while performing the load and then enable them, in which case, having a unique index on the `PRIMARY KEY` will still cause the data to be verified during the load. Therefore, you can first create a nonunique index on the column designated as `PRIMARY KEY` and then create the `PRIMARY KEY` column and specify that it should use the existing index. The following examples illustrate this process:

Step 1: Create the table:

```
CREATE TABLE NEW_EMP2

 (employee_id NUMBER(6),
 first_name  VARCHAR2(20),
 last_name   VARCHAR2(25)

 );
```

Step 2: Create the index:

```
CREATE INDEX emp_id_idx2 ON
  new_emp2(employee_id);
```

Step 3: Create the `PRIMARY KEY`:

```
ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id) USING INDEX
 emp_id_idx2;
```

Oracle Internal & Oracle Academy Use Only

Function-Based Indexes

- A function-based index is based on expressions.
- The index expression is built from table columns, constants, SQL functions, and user-defined functions.

```
CREATE INDEX upper_dept_name_idx
ON dept2(UPPER(department_name));
```

```
Index UPPER_DEPT_NAME_IDX created.
```

```
SELECT *
FROM   dept2
WHERE  UPPER(department_name) = 'SALES';
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	80	Sales	145	2500

1 row fetched in 0.012 seconds

*Note: The time may differ for you.

31

Function-based indexes defined with the `UPPER(column_name)` or `LOWER(column_name)` keywords allow non-case-sensitive searches. For example, consider the following index:

```
CREATE INDEX upper_last_name_idx ON emp2 (UPPER(last_name));
```

This facilitates processing queries such as:

```
SELECT * FROM emp2 WHERE UPPER(last_name) = 'KING';
```

The Oracle Server uses the index only when that particular function is used in a query. For example, the following statement may use the index; however, without the `WHERE` clause, the Oracle Server may perform a full table scan:

```
SELECT   *
FROM     employees
WHERE    UPPER (last_name) IS NOT NULL
ORDER BY UPPER (last_name);
```

Note: For creating a function-based index, you need the `QUERY REWRITE` system privilege. The `QUERY_REWRITE_ENABLED` initialization parameter must be set to `TRUE` for a function-based index to be used.

The Oracle Server treats indexes with columns marked `DESC` as function-based indexes. The columns marked `DESC` are sorted in descending order.

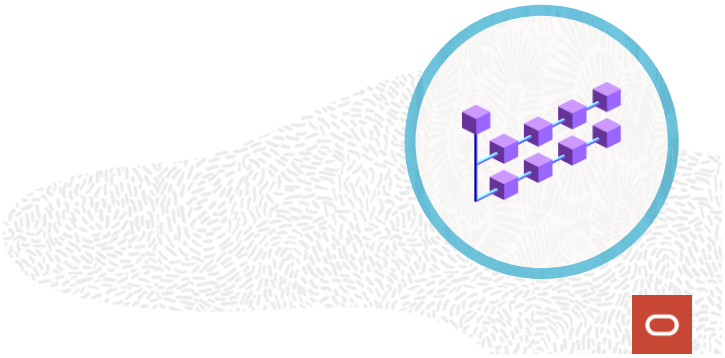
Observe that each time you run the same query on an index-based table, the time required to fetch the row(s) reduces.

Oracle Internal & Oracle Academy Use Only

Creating Multiple Indexes on the Same Set of Columns

You can create multiple indexes on the same set of columns if the indexes:

- Are of different types
- Use different partitioning
- Have different uniqueness properties



32

You can create multiple indexes on the same set of columns if the indexes are of different types, use different partitioning, or have different uniqueness properties. For example, you can create a B-tree index and a bitmap index on the same set of columns.

Similarly, you can create a unique and non-unique index on the same set of columns.

When you have multiple indexes on the same set of columns, only one of these indexes can be visible at a time.

Oracle Internal & Oracle Academy Use Only

Creating Multiple Indexes on the Same Set of Columns: Example

1

```
CREATE INDEX emp_id_name_ix1
ON employees(employee_id, first_name);
```

```
Index EMP_ID_NAME_IX1 created.
```

2

```
ALTER INDEX emp_id_name_ix1 INVISIBLE;
```

```
Index EMP_ID_NAME_IX1 altered.
```

3

```
CREATE BITMAP INDEX emp_id_name_ix2
ON employees(employee_id, first_name);
```

```
Bitmap index EMP_ID_NAME_IX2 created.
```

33

1. The code example shows the creation of a B-tree index, `emp_id_name_ix1`, on the `employee_id` and `first_name` column of the employees table in the `HR` schema. By default, Oracle Server creates a B-tree index. In a B-tree, you traverse the branches until you get to the node that has the data you are searching for.
2. After the creation of the index, it is altered to make it invisible.
3. Then a bitmap index is created on the `employee_id` and `first_name` column of the `employees` table in the `HR` schema. The bitmap index, `emp_id_name_ix2`, is visible by default.

Oracle Internal & Oracle Academy Use Only

Index Information

- `USER_INDEXES` provides information about your indexes.
- `USER_IND_COLUMNS` describes columns of indexes owned by you and columns of indexes on your tables.

```
DESCRIBE user_indexes
```

```
Name                    Null     Type
----------------------- -------- --------------
INDEX_NAME              NOT NULL VARCHAR2(128)
INDEX_TYPE                       VARCHAR2(27)
TABLE_OWNER             NOT NULL VARCHAR2(128)
TABLE_NAME              NOT NULL VARCHAR2(128)
TABLE_TYPE                       VARCHAR2(11)
UNIQUENESS                       VARCHAR2(9)
COMPRESSION                      VARCHAR2(13)
PREFIX_LENGTH                    NUMBER
TABLESPACE_NAME                  VARCHAR2(30)
...
```

You query the `USER_INDEXES` view to find out the names of your indexes, the table name on which the index is created, and whether the index is unique.

Note: For a complete listing and description of the columns in the `USER_INDEXES` view, see "`USER_INDEXES`" in *Oracle® Database Reference 19c*.

Oracle Internal & Oracle Academy Use Only

USER_INDEXES: Examples

```
SELECT index_name, table_name, uniqueness
FROM   user_indexes
WHERE  table_name = 'EMPLOYEES';
```

1

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
2	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
3	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
4	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
5	EMP_NAME_IX	EMPLOYEES	NONUNIQUE

...

```
SELECT index_name, table_name
FROM   user_indexes
WHERE  table_name = 'EMP_LIB';
```

2

	INDEX_NAME	TABLE_NAME
1	SYS_C0010979	EMP_LIB

35

In example **1** in the slide, the USER_INDEXES view is queried to find the name of the index, name of the table on which the index is created, and whether the index is unique.

In example **2** in the slide, observe that the Oracle Server gives a generic name to the index that is created for the PRIMARY KEY column. The EMP_LIB table is created by using the following code:

```
CREATE TABLE emp_lib
   (book_id NUMBER(6) PRIMARY KEY,
    title VARCHAR2(25),
    category VARCHAR2(20));
```

Oracle Internal & Oracle Academy Use Only

Querying USER_IND_COLUMNS

```
DESCRIBE user_ind_columns
```

```
Name            Null Type
--------------- ---- --------------
INDEX_NAME           VARCHAR2(128)
TABLE_NAME           VARCHAR2(128)
COLUMN_NAME          VARCHAR2(4000)
COLUMN_POSITION      NUMBER
COLUMN_LENGTH        NUMBER
CHAR_LENGTH          NUMBER
DESCEND              VARCHAR2(4)
```

```
SELECT index_name, column_name, table_name
FROM   user_ind_columns
WHERE  index_name = 'LNAME_IDX';
```

	INDEX_NAME	COLUMN_NAME	TABLE_NAME
1	LNAME_IDX	LAST_NAME	EMP_TEST

36

The USER_IND_COLUMNS dictionary view provides information, such as the name of the index, name of the indexed table, name of a column within the index, and the column's position within the index.

For the example in the slide, the emp_test table and LNAME_IDX index are created by using the following code:

```
CREATE TABLE emp_test AS SELECT * FROM employees;

CREATE INDEX lname_idx ON emp_test(last_name);
```

Oracle Internal & Oracle Academy Use Only

Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command:

```
DROP INDEX index;
```

- Remove the `emp_last_name_idx` index from the data dictionary:

```
DROP INDEX emp_last_name_idx;

Index EMP_LAST_NAME_IDX dropped.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

37

You cannot modify indexes. To change an index, you must drop it and then re-create it.

You can remove an index definition from the data dictionary by issuing the `DROP INDEX` statement. To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

In the syntax, *`index`* is the name of the index.

You can drop an index by using the `ONLINE` keyword:

```
DROP INDEX emp_indx ONLINE;
```

Specify `ONLINE` to indicate that DML operations on the table are allowed while dropping the index.

Note: If you drop a table, indexes and constraints are automatically dropped, but views remain.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Automatically generate sequence numbers by using a sequence generator
- Use synonyms to provide alternative names for objects
- Create indexes to improve the speed of query retrieval
- Find information about your objects through the following dictionary views:
  - `USER_INDEXES`
  - `USER_SEQUENCES`
  - `USER_SYNONYMS`

38

In this lesson, you should have learned about database objects, such as sequences, indexes, and synonyms.

Oracle Internal & Oracle Academy Use Only

Practice 13: Overview

This practice covers the following topics:

- Creating sequences
- Using sequences
- Querying the dictionary views for sequence information
- Creating synonyms
- Querying the dictionary views for synonyms information
- Creating indexes
- Querying the dictionary views for indexes information

39

This lesson's practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym. You also learn how to query the data dictionary views for sequence, synonyms, and index information.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

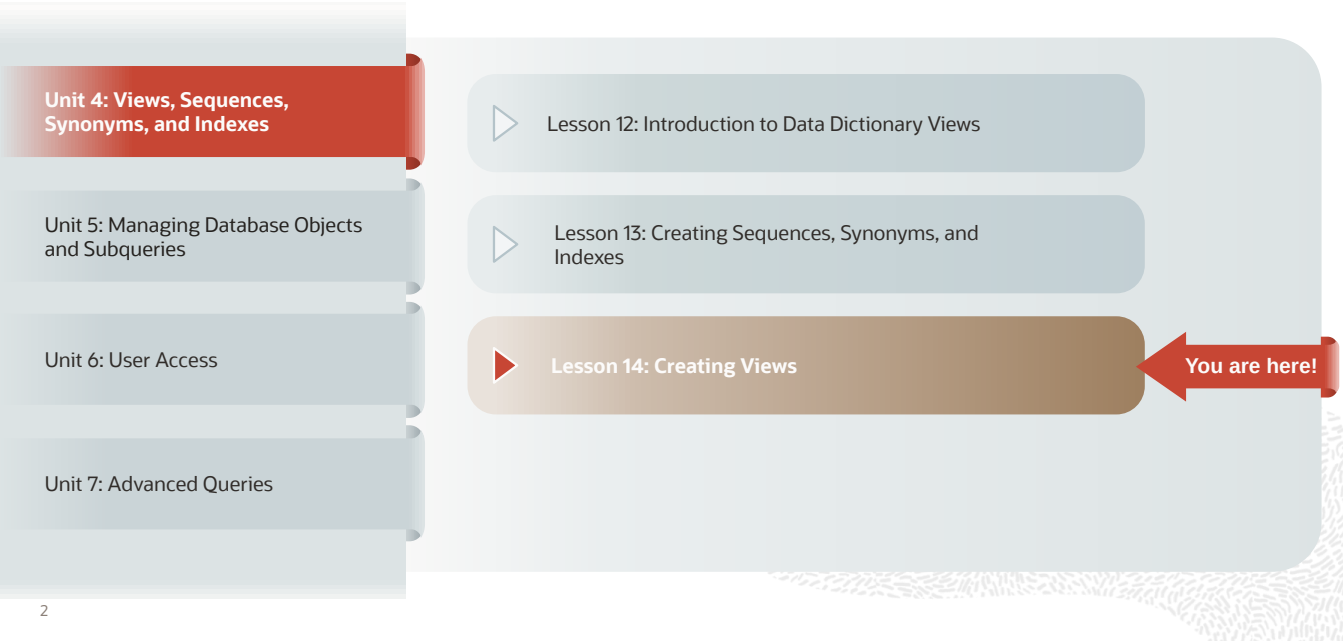
ORACLE

14

Creating Views

Oracle Internal & Oracle Academy Use Only

Course Roadmap



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

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Create simple and complex views
- Retrieve data from views
- Query dictionary views for view information

3

In this lesson, you are introduced to views, and you learn the basics of creating and using views.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

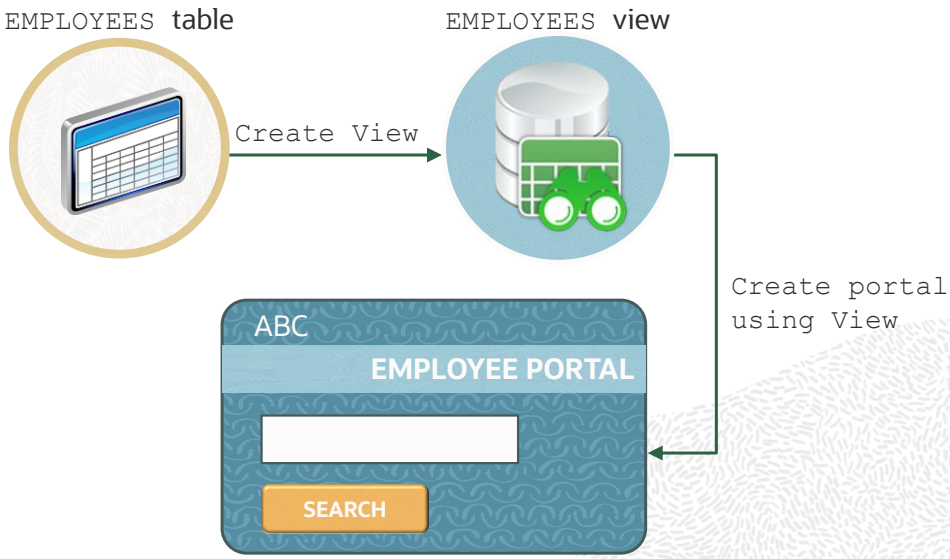
- Overview of views
- Creating, modifying, and retrieving data from a view
- Data manipulation language (DML) operations on a view
- Dropping a view

4

This section gives an overview of views.

Oracle Internal & Oracle Academy Use Only

Why Views?



5

ABC is a company, which consists of around 1000 employees. The HR department of the company maintains a database to store all the employee information, such as full name, date of birth, department, salary, manager, designation, phone number, hire date, and so on. This information is confidential and cannot be accessed by all the employees.

The company now decides to create an employee search portal for the use of employees. This portal should display only the necessary information (such as name, department, and manager) and should *not* display any confidential information (such as salary). To achieve this, the company need not create a separate table, which stores the same data with fewer columns.

Instead, a view is created with the columns required based on the underlying `EMPLOYEES` table. This helps in removing redundancy and hiding sensitive information securely. You learn more about views and how to use them in the following slides.

Oracle Internal & Oracle Academy Use Only

Database Objects

Object	Description
Table	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 data retrieval queries
Synonym	Gives alternative names to objects

6

There are several other objects in a database in addition to tables.

With views, you can present and hide data from the tables.

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of data retrieval queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

Oracle Internal & Oracle Academy Use Only

What Is a View?

EMPLOYEES table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-11	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-09	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-09	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-14	IT_PROG	9000
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-15	IT_PROG	6000
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-13	IT_PROG	4800
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-14	IT_PROG	4800
[illegible]	[illegible]	[illegible]	[illegible]	[illegible].5567	07-FEB-15	IT_PROG	4200
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]4.4569	17-AUG-10	FI_MGR	12008
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]4.4169	16-AUG-10	FI_ACCOUNT	9000
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]4.4269	28-SEP-13	FI_ACCOUNT	8200
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]4.4309	30-SEP-13	FI_ACCOUNT	7700
[illegible]	[illegible]	[illegible]	[illegible]	[illegible].4469	07-MAR-14	FI_ACCOUNT	7800
113	Luis	Popp	LPUPP	515.124.4567	07-DEC-15	FI_ACCOUNT	6900
114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-10	PU_MAN	11000
115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-11	PU_CLERK	3100

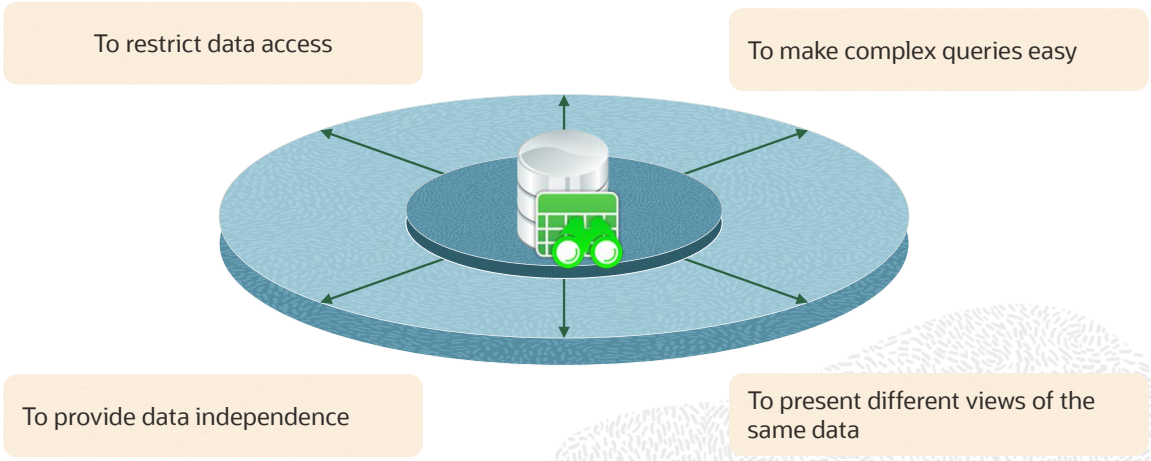
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
101	Neena	Kochhar	17000
102	Lex	De Haan	17000
103	Alexander	Hunold	9000
104	Bruce	Ernst	6000

7

You can present logical subsets or combinations of data by creating views of tables. A view is a schema object and is stored as a `SELECT` statement based on a table or another view. A view contains no data of its own, but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called *base tables*. The view is stored as a `SELECT` statement in the data dictionary.

Oracle Internal & Oracle Academy Use Only

Advantages of Views



8

The following are some of the advantages of views:

- Views restrict access to data because they display selected columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

For more information, see the "`CREATE VIEW`" section in *Oracle Database SQL Language Reference* for Oracle Database 19*c*.

Oracle Internal & Oracle Academy Use Only

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

9

There are two classifications for views: simple and complex. The basic difference is related to the DML (`INSERT`, `UPDATE`, and `DELETE`) operations.

- A simple view:
  - Derives data from only one table
  - Contains no functions or groups of data
  - Usually performs DML operations through the view
- A complex view:
  - Derives data from many tables
  - Contains functions or groups of data
  - Does not always allow DML operations through the view

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Overview of views
- Creating, modifying, and retrieving data from a view
- Data manipulation language (DML) operations on a view
- Dropping a view

10

In this section, you learn how to create, modify, and retrieve data from a view.

Oracle Internal & Oracle Academy Use Only

Creating a View

- You embed a subquery in the `CREATE VIEW` statement:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- The subquery can contain complex `SELECT` syntax.

11

You can create a view by embedding a subquery in the `CREATE VIEW` statement.
In the syntax:

`OR REPLACE`	Re-creates the view if it already exists. You can use this clause to change the definition of an existing view without dropping, re-creating, and regranting object privileges previously granted on it.
`FORCE`	Creates the view regardless of whether or not the base tables exist
`NOFORCE`	Creates the view only if the base tables exist (This is the default.)
`view`	Is the name of the view
`alias`	Specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)
`subquery`	Is a complete `SELECT` statement (You can use aliases for the columns in the `SELECT` list.)
`WITH CHECK OPTION`	Specifies that only those rows that are accessible to the view can be inserted or updated
`Constraint`	Is the name assigned to the `CHECK OPTION` constraint
`WITH READ ONLY`	Ensures that no DML operations can be performed on this view

Note: In SQL Developer, click the Run Script icon or press F5 to run the data definition language (DDL) statements. The feedback messages will be shown on the Script Output tabbed page.

Oracle Internal & Oracle Academy Use Only

Creating a View

- Create the `EMPVU80` view, which contains details of the employees in department 80:

```
CREATE VIEW  empvu80
 AS SELECT  employee_id, last_name, salary
    FROM    employees
    WHERE   department_id = 80;
```

```
View EMPVU80 created.
```

- Describe the structure of the view by using the SQL*Plus `DESCRIBE` command:

```
DESCRIBE empvu80;
```

12

The example in the slide creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the `DESCRIBE` command.

```
Name        Null     Type
----------- -------- ------------
EMPLOYEE_ID NOT NULL NUMBER(6)
LAST_NAME   NOT NULL VARCHAR2(25)
SALARY               NUMBER(8,2)
```

Guidelines

- The subquery that defines a view can contain complex `SELECT` syntax, including joins, groups, and subqueries.
- If you do not specify a constraint name for the view created with the `WITH CHECK OPTION`, the system assigns a default name in the `SYS_C`*n* format.
- You can use the `OR REPLACE` option to change the definition of the view without dropping and re-creating it, or re-granting the object privileges previously granted on it.

Oracle Internal & Oracle Academy Use Only

Creating a View

- Create a view by using column aliases in the subquery:

```
CREATE VIEW  salvu50
 AS SELECT  employee_id ID_NUMBER, last_name NAME,
            salary*12 ANN_SALARY
    FROM    employees
    WHERE   department_id = 50;
```

```
View SALVU50 created.
```

- Select the columns from this view by the given alias names.

```
SELECT ID_NUMBER, NAME, ANN_SALARY
FROM salvu50;
```

13

You can control the column names by including column aliases in the subquery.

The example in the slide creates a view containing the employee number (`EMPLOYEE_ID`) with the alias `ID_NUMBER`, name (`LAST_NAME`) with the alias `NAME`, and annual salary (`SALARY`) with the alias `ANN_SALARY` for every employee in department 50.

Alternatively, you can use an alias after the `CREATE` statement and before the `SELECT` subquery. The number of aliases listed must match the number of expressions selected in the subquery.

```
CREATE OR REPLACE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY)
  AS SELECT  employee_id, last_name, salary*12
     FROM    employees
     WHERE   department_id = 50;
```

Oracle Internal & Oracle Academy Use Only

Retrieving Data from a View

```
SELECT *
FROM   salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	120	Weiss	96000
2	121	Fripp	98400
3	122	Kaufling	94800
4	123	Vollman	78000
5	124	Mourgos	69600
6	125	Nayer	38400
...7	126	Mikkilineni	32400

14

You can retrieve data from a view as you would from any table. You can display either the contents of the entire view or just specific rows and columns.

Oracle Internal & Oracle Academy Use Only

Modifying a View

- Modify the EMPVU80 view by using a `CREATE OR REPLACE VIEW` clause. Add an alias for each column name:

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
   FROM    employees
   WHERE   department_id = 80;
```

```
View EMPVU80 created.
```

- Column aliases in the `CREATE OR REPLACE VIEW` clause are listed in the same order as the columns in the subquery.

15

With the `REPLACE` option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and re-granting object privileges.

Note: When assigning column aliases in the `CREATE OR REPLACE VIEW` clause, remember that the aliases are listed in the same order as the columns in the subquery.

Oracle Internal & Oracle Academy Use Only

Creating a Complex View

Create a complex view that contains group functions to display values from two tables:

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
            MAX(e.salary),AVG(e.salary)
   FROM     employees e JOIN departments d
   USING    (department_id)
   GROUP BY d.department_name;
```

```
View DEPT_SUM_VU created.
```

16

The example in the slide creates a complex view of department names, minimum salaries, maximum salaries, and the average salaries by department. Note that alternative names have been specified for the view. This is a requirement if any column of the view is derived from a function or an expression.

You can view the structure of the view by using the `DESCRIBE` command. Display the contents of the view by issuing a `SELECT` statement.

```
SELECT   *
FROM     dept_sum_vu;
```

Oracle Internal & Oracle Academy Use Only

View Information

```
DESCRIBE user_views
```

```
Name              Null     Type
----------------- -------- --------------
VIEW_NAME         NOT NULL VARCHAR2(128)
TEXT_LENGTH                NUMBER
TEXT                       LONG
TEXT_VC                    VARCHAR2(4000)
TYPE_TEXT_LENGTH           NUMBER
TYPE_TEXT                  VARCHAR2(4000)
...
```

```
SELECT view_name FROM user_views;
```

	VIEW_NAME
1	EMP_DETAILS_VIEW
2	SALVU50
3	EMPVU80
4	DEPT_SUM_VU

```
SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';
```

	TEXT
1	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.co

```
...
AND c.region_id = r.region_id AND j.job_id = e.job_idWITH READ ONLY
```

17

After your view is created, you can query the data dictionary view called `USER_VIEWS` to see the name of the view and the view definition.

The text of the `SELECT` statement that constitutes your view is stored in a `LONG` column. The `TEXT_LENGTH` column is the number of characters in the `SELECT` statement. By default, when you select from a `LONG` column, only the first 80 characters of the column's value are displayed. To see more than 80 characters in SQL*Plus, use the `SET LONG` command:

```
SET LONG 1000
```

In the examples in the slide:

1. The `USER_VIEWS` columns are displayed. Note that this is a partial listing.
2. The names of your views are retrieved
3. The `SELECT` statement for the `EMP_DETAILS_VIEW` is displayed from the dictionary

Data Access Using Views

When you access data by using a view, the Oracle Server performs the following operations:

- It retrieves the view definition from the data dictionary table `USER_VIEWS`.
- It checks access privileges for the view base table.
- It converts the view query into an equivalent operation on the underlying base table or tables. That is, data is retrieved from, or an update is made to, the base tables.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Overview of views
- Creating, modifying, and retrieving data from a view
- Data manipulation language (DML) operations on a view
- Dropping a view

18

In this section, you will learn how to perform DML operations on a view.

Oracle Internal & Oracle Academy Use Only

Rules for Performing DML Operations on a View

- You can usually perform `DML` operations on simple views.
- You cannot remove a row if the view contains:
  - Group functions
  - A `GROUP` BY clause
  - The `DISTINCT` keyword
  - The pseudocolumn `ROWNUM` keyword

19

- You can perform DML operations on data in a view if those operations follow certain rules.
- You can remove a row from a view unless it contains any of the following:
  - Group functions
  - A `GROUP BY` clause
  - The `DISTINCT` keyword
  - The `pseudocolumn` ROWNUM keyword

Oracle Internal & Oracle Academy Use Only

Rules for Performing Modify Operations on a View

You cannot modify data in a view if it contains:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Expressions

20

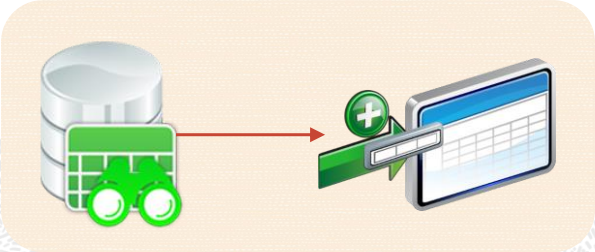
You can modify data in a view unless it contains any of the conditions mentioned in the slide.

Oracle Internal & Oracle Academy Use Only

Rules for Performing Insert Operations Through a View

You cannot add data in a view if the view includes:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Columns defined by expressions
- `NOT NULL` columns without default value in the base tables that are not selected by the view



21

You can add data through a view unless it contains any of the items listed in the slide. You cannot add data to a view if the view contains NOT NULL columns without default values in the base table. All the required values must be present in the view. Remember that you are adding values directly to the underlying table through the view.

For more information, see the “CREATE VIEW” section in Oracle Database SQL Language.

Oracle Internal & Oracle Academy Use Only

Using the WITH CHECK OPTION Clause

```
CREATE OR REPLACE VIEW empvu20
AS SELECT *
   FROM     employees
   WHERE    department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```

```
View EMPVU20 created.
```

22

It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows that the view cannot select. Therefore, it enables integrity constraints and data validation checks to be enforced on data being inserted or updated. If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, along with the constraint name if that has been specified.

```
UPDATE empvu20
SET    department_id = 10
WHERE  employee_id = 201;
```

Error:

```
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 - "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

Note: No rows are updated because, if the department number were to change to 10, the view would no longer be able to see that employee. With the WITH CHECK OPTION clause, therefore, the view can see only the employees in department 20 and does not allow the department number for those employees to be changed through the view.

Oracle Internal & Oracle Academy Use Only

Denying DML Operations

- You can ensure that no `DML` operations occur on your view by adding the `WITH READ ONLY` option to your view definition.
- Any attempt to perform a `DML` operation on any row in the view results in an Oracle server error.

23

The example in the following slide modifies the EMPVU10 view to prevent any DML operations on the view.

Oracle Internal & Oracle Academy Use Only

Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
    (employee_number, employee_name, job_title)
AS SELECT employee_id, last_name, job_id
   FROM     employees
   WHERE    department_id = 10
   WITH READ ONLY ;
```

```
View EMPVU10 created.
```

24

Any attempt to remove a row from a view with a read-only constraint results in an error:

```
DELETE FROM empvu10
WHERE  employee_number = 200;
```

Similarly, any attempt to insert a row or modify a row using the view with a read-only constraint results in the same error.

```
Error report:
SQL Error: ORA-42399: cannot perform a DML operation on a read-only view
```

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Overview of views
- Creating, modifying, and retrieving data from a view
- Data manipulation language (DML) operations on a view
- Dropping a view

25

In this section, you learn how to drop a view.

Oracle Internal & Oracle Academy Use Only

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

Syntax:

```
DROP VIEW view;
```

Example:

```
DROP VIEW empvu80;
```

```
View EMPVU80 dropped.
```

26

You use the `DROP VIEW` statement to remove a view. The statement removes the view definition from the database. However, dropping views has no effect on the tables on which the view was based. Alternatively, views or other applications based on the deleted views become invalid. Only the creator or a user with the `DROP ANY VIEW` privilege can remove a view.

In the syntax, *`view`* is the name of the view.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Create, modify, and remove views
- Query the dictionary views for view information

27

In this lesson, you should have learned about views.

Oracle Internal & Oracle Academy Use Only

Practice 14: Overview

This practice covers the following topics:

- Creating a simple view
- Creating a complex view
- Creating a view with a check constraint
- Attempting to modify data in the view
- Querying the dictionary views for view information
- Removing views

28

The practice provides you with a variety of exercises in creating, using, querying data dictionary views for view information, and removing views.

Oracle Internal & Oracle Academy Use Only

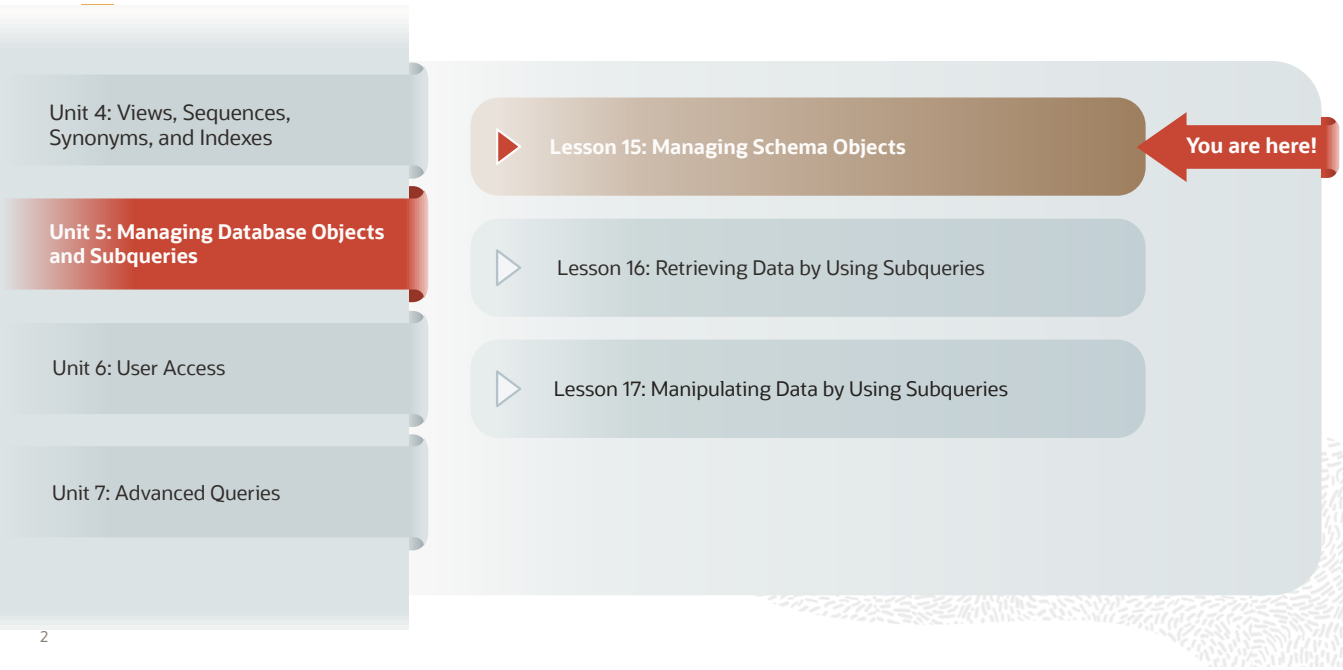
ORACLE

15

Managing Schema Objects

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 5, you will learn to use SQL statements to query and display data from multiple tables using Joins, 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:

- Manage constraints
- Create and use temporary tables
- Create and use external tables

3

In this lesson, you learn about constraints and altering existing objects. You will also learn about external tables and the provision to name the index at the time of creating a `PRIMARY KEY` constraint.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Managing constraints:
  - Adding and dropping a constraint
  - Enabling and disabling a constraint
  - Deferring constraints
- Creating and using temporary tables
- Creating and using external tables

4

This section discusses how to manage constraints. You learn to add, drop, enable, disable, and defer a constraint.

Oracle Internal & Oracle Academy Use Only

Adding a Constraint Syntax

Use the `ALTER TABLE` statement to:

- Add or drop a constraint, but not to modify its structure
- Enable or disable constraints
- Add a `NOT NULL` constraint by using the `MODIFY` clause

```
ALTER TABLE   <table_name>
ADD [CONSTRAINT <constraint_name>]
type (<column_name>);
```

5

You can add a constraint for existing tables by using the `ALTER TABLE` statement with the `ADD` clause.

In the syntax:

table_name	Is the name of the table
constraint_name	Is the name of the constraint
type	Is the constraint type
column_name	Is the name of the column affected by the constraint

The constraint name syntax is optional, although recommended. If you do not name your constraints, the system generates constraint names.

Guidelines

- You can add, drop, enable, or disable a constraint, but you cannot modify its structure.
- You can add a `NOT NULL` constraint to an existing column by using the `MODIFY` clause of the `ALTER TABLE` statement.

Note: You can define a `NOT NULL` column only if the table is empty or if the column has a value for every row.

Oracle Internal & Oracle Academy Use Only

Adding a Constraint

Add a `FOREIGN KEY` constraint to the `EMP2` table indicating that a manager must already exist as a valid employee in the `EMP2` table.

```
ALTER TABLE emp2
MODIFY employee_id PRIMARY KEY;
```

```
Table EMP2 altered.
```

```
ALTER TABLE emp2
ADD CONSTRAINT emp_mgr_fk
   FOREIGN KEY(manager_id)
   REFERENCES emp2(employee_id);
```

```
Table EMP2 altered.
```

6

The first example in the slide modifies the `EMP2` table to add a `PRIMARY KEY` constraint on the `EMPLOYEE_ID` column. Note that because no constraint name is provided, the constraint is automatically named by the Oracle Server.

The second example in the slide creates a `FOREIGN KEY` constraint on the `EMP2` table. The constraint ensures that a manager exists as a valid employee in the `EMP2` table.

Note: The `EMP2` table is created with the following statement:

```
CREATE TABLE emp2 as
SELECT * FROM employees;
```

Oracle Internal & Oracle Academy Use Only

Dropping a Constraint

- The drop_constraint_clause enables you to drop an integrity constraint from a database.
- Remove the manager constraint from the `EMP2` table:

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
```

```
Table EMP2 altered.
```

- Remove the `PRIMARY KEY` constraint on the `EMP2` table and drop the associated `FOREIGN KEY` constraint on the `EMP2.MANAGER_ID` column:

```
ALTER TABLE emp2
DROP PRIMARY KEY CASCADE;
```

```
Table EMP2 altered.
```

7

To drop a constraint, you can identify the constraint name from the `USER_CONSTRAINTS` and `USER_CONS_COLUMNS` data dictionary views. Then use the `ALTER TABLE` statement with the `DROP` clause. The `CASCADE` option of the `DROP` clause causes any dependent constraints also to be dropped.

Syntax

```
ALTER TABLE        table
 DROP  PRIMARY KEY | UNIQUE (column) |
       CONSTRAINT   constraint  [CASCADE];
```

In the syntax:

table	Is the name of the table
column	Is the name of the column affected by the constraint
constraint	Is the name of the constraint

When you drop an integrity constraint, that constraint is no longer enforced by the Oracle Server and is no longer available in the data dictionary.

Oracle Internal & Oracle Academy Use Only

Dropping a Constraint ONLINE

You can specify the ONLINE keyword to indicate that DML operations on the table are allowed while dropping the constraint.

```
ALTER TABLE myemp2
DROP CONSTRAINT emp_name_pk ONLINE;
```

```
Table MYEMP2 altered.
```

DML operations allowed while dropping a constraint

8

You can also drop a constraint by using an ONLINE keyword:

```
ALTER TABLE myemp2
DROP CONSTRAINT emp_id_pk ONLINE;
```

Use the ALTER TABLE statement with the DROP clause. The ONLINE option of the DROP clause indicates that DML operations are allowed on the table while dropping the constraint.

Note: The myemp2 table is created using the following statement:

```
CREATE TABLE myemp2
(emp_id NUMBER(6) CONSTRAINT emp_name_pk PRIMARY KEY ,
emp_name VARCHAR2(20));
```

Oracle Internal & Oracle Academy Use Only

ON DELETE Clause

- Use the `ON DELETE CASCADE` clause to delete child rows when a parent key is deleted:

```
ALTER TABLE dept2 ADD CONSTRAINT dept_lc_fk
FOREIGN KEY (location_id)
REFERENCES locations(location_id) ON DELETE CASCADE;
```

```
Table DEPT2 altered.
```

- Use the `ON DELETE SET NULL` clause to set the child rows value to null when a parent key is deleted:

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk
FOREIGN KEY (department_id)
REFERENCES departments(department_id) ON DELETE SET NULL;
```

```
Table EMP2 altered.
```

9

ON DELETE

By using the `ON DELETE` clause, you can determine how Oracle Database handles referential integrity if you remove a referenced primary or unique key value.

ON DELETE CASCADE

The `ON DELETE CASCADE` clause allows parent key data that is referenced from the child table to be deleted, but not updated. When data in the parent key is deleted, all the rows in the child table that depend on the deleted parent key values are also deleted. To specify this referential action, include the `ON DELETE CASCADE` clause in the definition of the `FOREIGN KEY` constraint.

ON DELETE SET NULL

When data in the parent key is deleted, the `ON DELETE SET NULL` clause causes all the rows in the child table that depend on the deleted parent key value to be converted to null.

If you omit this clause, Oracle does not allow you to delete referenced key values in the parent table that have dependent rows in the child table.

Oracle Internal & Oracle Academy Use Only

Cascading Constraints

The `CASCADE CONSTRAINTS` clause:

- Is used along with the `DROP COLUMN` clause
- Drops all referential integrity constraints that refer to the `PRIMARY` and `UNIQUE` keys defined on the dropped columns
- Drops all multicolumn constraints defined on the dropped columns

This statement illustrates the usage of the `CASCADE CONSTRAINTS` clause. Assume that the `TEST1` table is created as follows:

```
CREATE TABLE test1 (
  col1_pk NUMBER PRIMARY KEY,
  col2_fk NUMBER,
  col1 NUMBER,
  col2 NUMBER,
  CONSTRAINT fk_constraint FOREIGN KEY (col2_fk) REFERENCES
    test1,
  CONSTRAINT ck1 CHECK (col1_pk > 0 and col1 > 0),
  CONSTRAINT ck2 CHECK (col2_fk > 0));
```

An error is returned for the following statements:

- `ALTER TABLE test1 DROP (col1_pk);`

`ERROR: col1_pk` is a parent key.

- `ALTER TABLE test1 DROP (col1);`

`ERROR: col1` is referenced by the multicolumn constraint, `ck1`.

Oracle Internal & Oracle Academy Use Only

Cascading Constraints

Example:

```
ALTER TABLE emp2
DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

```
Table EMP2 altered.
```

```
ALTER TABLE test1
DROP (col1_pk, col2_fk, col1) CASCADE CONSTRAINTS;
```

```
Table TEST1 altered.
```

11

Submitting the following statement drops the `EMPLOYEE_ID` column, the `PRIMARY KEY` constraint, and any `FOREIGN KEY` constraints referencing the `PRIMARY KEY` constraint for the `EMP2` table:

```
ALTER TABLE emp2 DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

If all columns referenced by the constraints defined on the dropped columns are also dropped, `CASCADE CONSTRAINTS` is not required. For example, assuming that no other referential constraints from other tables refer to the `COL1_PK` column, it is valid to submit the following statement without the `CASCADE CONSTRAINTS` clause for the `TEST1` table created on the previous page:

```
ALTER TABLE test1 DROP (col1_pk, col2_fk, col1);
```

Guidelines

- Enabling a `PRIMARY KEY` constraint that was disabled with the `CASCADE` option does not enable any `FOREIGN KEYs` that are dependent on `PRIMARY KEY`.
- To enable a `UNIQUE` or `PRIMARY KEY` constraint, you must have the privileges necessary to create an index on the table.

Oracle Internal & Oracle Academy Use Only

Renaming Table Columns and Constraints

- Use the `RENAME` table clause of the `ALTER TABLE` statement to rename tables.

```
ALTER TABLE marketing RENAME to new_marketing;
```

1

- Use the `RENAME COLUMN` clause of the `ALTER TABLE` statement to rename table columns.

```
ALTER TABLE new_marketing RENAME COLUMN team_id
TO id;
```

2

- Use the `RENAME CONSTRAINT` clause of the `ALTER TABLE` statement to rename any existing constraint for a table.

```
ALTER TABLE new_marketing RENAME CONSTRAINT mktg_pk
TO new_mktg_pk;
```

3

12

The `RENAME` table clause enables you to rename an existing table in any schema (except the `SYS` schema).To rename a table, you must either be the database owner or the table owner.

When you rename a table column, the new name must not conflict with the name of any existing column in the table. You cannot use any other clauses in conjunction with the `RENAME COLUMN` clause.

The examples in the slide use the `marketing` table with the `PRIMARY KEY` `mktg_pk` defined on the `id` column.

```
CREATE TABLE marketing (team_id  NUMBER(10),
                        target  VARCHAR2(50),
CONSTRAINT mktg_pk PRIMARY KEY(team_id));
```

Example **1** shows that the `marketing` table is renamed `new_marketing`.

Example **2** shows that the `team_id` column of the `new_marketing` table is renamed `id`.

Example **3** shows that the `mktg_pk` constraint is renamed `new_mktg_pk`.

When you rename any existing constraint for a table, the new name must not conflict with any of your existing constraint names. You can use the `RENAME CONSTRAINT` clause to rename system-generated constraint names.

Oracle Internal & Oracle Academy Use Only

Disabling Constraints

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.
- Apply the CASCADE option to disable the primary key. It will also disable all dependent FOREIGN KEY constraints automatically.

```
ALTER TABLE    emp2
DISABLE CONSTRAINT emp_dt_fk;
```

```
Table EMP2 altered.
```

```
ALTER TABLE    dept2
DISABLE primary key CASCADE;
```

```
Table DEPT2 altered.
```

13

You can disable a constraint, without dropping it or re-creating it, by using the ALTER TABLE statement with the DISABLE clause. You can also disable the primary key or unique key by using the CASCADE option.

Syntax

```
ALTER   TABLE   table
 DISABLE CONSTRAINT constraint [CASCADE];
```

In the syntax:

table	Is the name of the table
constraint	Is the name of the constraint

Guidelines

- You can use the DISABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.
- The CASCADE clause disables dependent integrity constraints.
- Disabling a UNIQUE or PRIMARY KEY constraint removes the unique index.

Oracle Internal & Oracle Academy Use Only

Enabling Constraints

- Activate an integrity constraint that is currently disabled in the table definition by using the ENABLE clause.

```
ALTER TABLE          emp2
ENABLE CONSTRAINT    emp_dt_fk;
```

```
Table EMP2 altered.
```

- A UNIQUE index is automatically created if you enable a UNIQUE key or a PRIMARY KEY constraint.

14

You can enable a constraint without dropping it or re-creating it by using the ALTER TABLE statement with the ENABLE clause.

Syntax

```
ALTER   TABLE      table
ENABLE  CONSTRAINT constraint;
```

In the syntax:

table	Is the name of the table
constraint	Is the name of the constraint

Guidelines

- If you enable a constraint, that constraint applies to all the data in the table. All the data in the table must comply with the constraint.
- If you enable a UNIQUE key or a PRIMARY KEY constraint, a UNIQUE or PRIMARY KEY index is created automatically. If an index already exists, it can be used by these keys.
- You can use the ENABLE clause in both the CREATE TABLE statement and the ALTER TABLE statement.

Oracle Internal & Oracle Academy Use Only

Constraint States

An integrity constraint defined on a table can be in one of the following states:

- ENABLE VALIDATE
- ENABLE NOVALIDATE
- DISABLE VALIDATE
- DISABLE NOVALIDATE

```
ALTER TABLE dept2
ENABLE NOVALIDATE PRIMARY KEY;
```

```
Table DEPT2 altered.
```

15

You can enable or disable integrity constraints at the table level using the CREATE TABLE or ALTER TABLE statement. You can also set constraints to VALIDATE or NOVALIDATE, in any combination with ENABLE or DISABLE, where:

- ENABLE ensures that all incoming data conforms to the constraint
- DISABLE allows incoming data, regardless of whether it conforms to the constraint
- VALIDATE ensures that existing data conforms to the constraint
- NOVALIDATE means that some existing data may not conform to the constraint

ENABLE VALIDATE is the same as ENABLE. The constraint is checked and is guaranteed to hold for all rows.

ENABLE NOVALIDATE means that the constraint is checked, but it does not have to be true for all rows. This allows existing rows to violate the constraint, while ensuring that all new or modified rows are valid. In an ALTER TABLE statement, ENABLE NOVALIDATE resumes constraint checking on disabled constraints without first validating all data in the table.

DISABLE NOVALIDATE is the same as DISABLE. The constraint is not checked and is not necessarily true.

DISABLE VALIDATE disables the constraint, drops the index on the constraint, and disallows any modification of the constrained columns.

Oracle Internal & Oracle Academy Use Only

Deferring Constraints

Constraints can have the following attributes:

- DEFERRABLE or NOT DEFERRABLE
- INITIALLY DEFERRED or INITIALLY IMMEDIATE

```
ALTER TABLE dept2
ADD CONSTRAINT dept2_id_pk
PRIMARY KEY (department_id)
DEFERRABLE INITIALLY DEFERRED;
```

Deferring constraint on creation

```
SET CONSTRAINTS dept2_id_pk IMMEDIATE;
```

Changing a specific constraint attribute

```
ALTER SESSION
SET CONSTRAINTS= IMMEDIATE;
```

Changing all constraints for a session

16

You can defer checking constraints for validity until the end of the transaction.

- A constraint is **deferred** if the system does not check whether the constraint is satisfied until a COMMIT statement is submitted. If a deferred constraint is violated, the database returns an error and the transaction is not committed and it is rolled back.
- If a constraint is **immediate** (not deferred), it is checked at the end of each statement. If it is violated, the statement is rolled back immediately.
- If a constraint causes an action (for example, DELETE CASCADE), that action is always taken as part of the statement that caused it, whether the constraint is deferred or immediate.
- Use the SET CONSTRAINTS statement to specify, for a particular transaction, whether a deferrable constraint is checked following each data manipulation language (DML) statement or when the transaction is committed. To create deferrable constraints, you must create a nonunique index for that constraint.
- You can define constraints as either DEFERRABLE or NOT DEFERRABLE (default) and either INITIALLY DEFERRED or INITIALLY IMMEDIATE (default). These attributes can be different for each constraint.

Usage scenario: The company policy dictates that department number 40 should be changed to 45. Changing the DEPARTMENT_ID column affects employees assigned to this department. Therefore, you make the PRIMARY KEY and FOREIGN KEYs deferrable and initially deferred. You update both department and employee information, and at the time of commit, all the rows are validated.

Oracle Internal & Oracle Academy Use Only

Difference Between INITIALLY DEFERRED and INITIALLY IMMEDIATE

INITIALLY DEFERRED	Waits until the transaction ends to check the constraint
INITIALLY IMMEDIATE	Checks the constraint at the end of the statement execution

```
CREATE TABLE emp_new_sal (salary NUMBER
        CONSTRAINT sal_ck CHECK (salary > 100)
        DEFERRABLE INITIALLY IMMEDIATE,
        bonus NUMBER
        CONSTRAINT bonus_ck CHECK (bonus > 0 )
        DEFERRABLE INITIALLY DEFERRED );
```

```
Table EMP_NEW_SAL created.
```

17

A constraint that is defined as deferrable can be specified as either INITIALLY DEFERRED or INITIALLY IMMEDIATE. The INITIALLY IMMEDIATE clause is the default.

In the example in the slide:

- The sal_ck constraint is created as DEFERRABLE INITIALLY IMMEDIATE
- The bonus_ck constraint is created as DEFERRABLE INITIALLY DEFERRED

After creating the emp_new_sal table, as shown in the slide, you attempt to insert values into the table and observe the results. When both the sal_ck and bonus_ck constraints are satisfied, the rows are inserted without an error.

Example 1: Insert a row that violates sal_ck. In the CREATE TABLE statement, sal_ck is specified as an initially immediate constraint. This means that the constraint is verified immediately after the INSERT statement and you observe an error.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

```
SQL Error: ORA-02290: check constraint (TEACH_B.SAL_CK) violated
02290. 00000 -  "check constraint (%s.%s) violated"
```

Example 2: Insert a row that violates bonus_ck. In the CREATE TABLE statement, bonus_ck is specified as deferrable and also initially deferred. Therefore, the constraint is not verified until you COMMIT or set the constraint state back to immediate.

```
1 row inserted
```

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

Oracle Internal & Oracle Academy Use Only

The row insertion is successful. But you observe an error when you commit the transaction.

```
COMMIT;
```

```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (TEACH_B.BONUS_CK) violated
02091. 00000 -  "transaction rolled back"
```

The commit failed due to constraint violation. Therefore, at this point, the transaction is rolled back by the database.

Example 3: Set the `DEFERRED` status to all constraints that can be deferred. Note that you can also set the `DEFERRED` status to a single constraint if required.

```
ALTER SESSION SET CONSTRAINTS = DEFERRED;
```

```
Session altered.
```

Now, if you attempt to insert a row that violates the `sal_ck` constraint, the statement is executed successfully.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

```
1 row inserted.
```

However, you observe an error when you commit the transaction. The transaction fails and is rolled back. This is because both the constraints are checked upon `COMMIT`.

```
COMMIT;
```

```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (TEACH_B.SAL_CK) violated
02091. 00000 -  "transaction rolled back"
```

Example 4: Set the `IMMEDIATE` status to both the constraints that were set as `DEFERRED` in the previous example.

```
ALTER SESSION SET CONSTRAINTS = IMMEDIATE;
```

```
Session altered.
```

You observe an error if you attempt to insert a row that violates either `sal_ck` or `bonus_ck`.

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

```
SQL Error: ORA-02290: check constraint (TEACH_B.BONUS_CK) violated
02290. 00000 -  "check constraint (%s.%s) violated"
```

Note: If you create a table without specifying constraint deferability, the constraint is checked immediately at the end of each statement. For example, with the `CREATE TABLE` statement of the `newemp_details` table, if you do not specify the `newemp_det_pk` constraint deferability, the constraint is checked immediately.

```
CREATE TABLE newemp_details(emp_id NUMBER,
emp_name VARCHAR2(20),
CONSTRAINT newemp_det_pk PRIMARY KEY(emp_id));
```

When you attempt to defer the `newemp_det_pk` constraint that is not deferrable, you observe the following error:

```
SET CONSTRAINT newemp_det_pk DEFERRED;
```

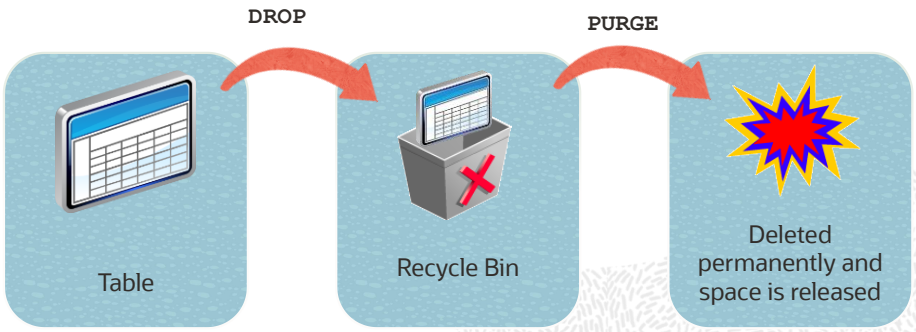
```
SQL Error: ORA-02447: cannot defer a constraint that is not deferrable
```

Oracle Internal & Oracle Academy Use Only

DROP TABLE … PURGE

```
DROP TABLE emp_new_sal PURGE;
```

```
Table EMP_NEW_SAL dropped.
```



19

Oracle Database provides a feature for dropping tables. When you drop a table, the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the `FLASHBACK TABLE` statement if you find that you dropped the table in error. If you want to immediately release the space associated with the table at the time you issue the `DROP TABLE` statement, then include the `PURGE` clause as shown in the statement in the slide.

Specify `PURGE` only if you want to drop the table and release the space associated with it in a single step. If you specify `PURGE`, the database does not place the table and its dependent objects into the recycle bin.

Using this clause is equivalent to first dropping the table and then purging it from the recycle bin. This clause saves you one step in the process. It also provides enhanced security if you want to prevent sensitive material from appearing in the recycle bin.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

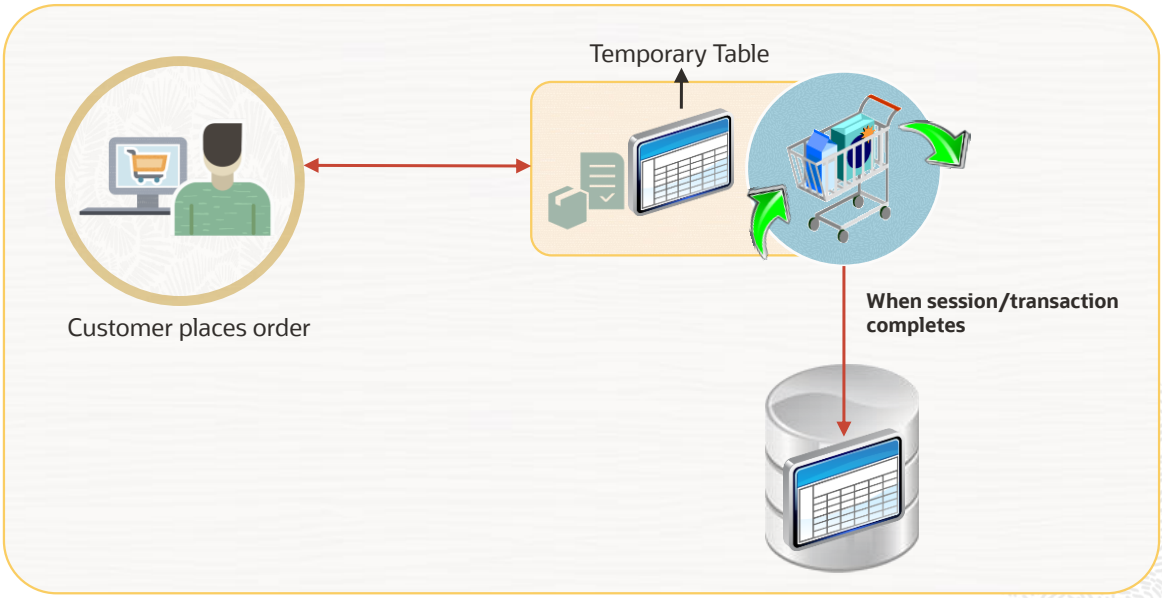
- Managing constraints:
  - Adding and dropping a constraint
  - Enabling and disabling a constraint
  - Deferring constraints
- Creating and using temporary tables
- Creating and using external tables

20

In this section, you learn to create and use temporary tables.

Oracle Internal & Oracle Academy Use Only

Using Temporary Tables



21

Brian is designing the OracleKart application. When customers place an order, the orders are to be stored in an `ORDER_DETAILS` table. Brian wants to enable customers, before they finalize their order, to use a “shopping cart” in which they can list items they wish to purchase. They should be able to add and remove items from this shopping cart. Storing this information in the `ORDER_DETAILS` table is not feasible because this table is not accessible to customers. Moreover, this data is not required to be stored permanently but only for the duration of a transaction. SQL enables you to store such information in temporary tables.

The shopping cart can be a temporary table. Each item is represented by a row in the temporary table. James continues to add and remove items from his cart before he finalizes on the items to buy. During this session, the cart data is private and the data in the temporary table keeps changing. After James finalizes his shopping and makes the payment, the application moves the rows from his cart to a permanent table, such as the `ORDER_DETAILS` table. At the end of the session, the data in the temporary table is automatically dropped.

Therefore, temporary tables are useful in applications where a result set must be buffered.

Oracle Internal & Oracle Academy Use Only

Temporary Table

A temporary table:

- Holds data that exists only for the duration of a transaction or session
  - Data is private to the session.
- Can be either a Global Temporary Table or a Private Temporary Table

22

A temporary table holds data that exists only for the duration of a transaction or session.

Data in a temporary table is private to the session. Each session can only see and modify its own data.

You can create either a global temporary table or a private temporary table.

Oracle Internal & Oracle Academy Use Only

Temporary Table Characteristics

Characteristic	Global	Private
Naming rules	Same as for permanent tables	Must be prefixed with ORA$PTT_
Visibility of table definition	All sessions	Only the session that created the table
Storage of table definition	Disk	Memory only
Types	Transaction-specific (ON COMMIT DELETE ROWS) or session-specific (ON COMMIT PRESERVE ROWS)	Transaction-specific (ON COMMIT DROP DEFINITION) or session-specific (ON COMMIT PRESERVE DEFINITION)

Oracle Internal & Oracle Academy Use Only

Creating a Global Temporary Table

1

```
CREATE GLOBAL TEMPORARY TABLE cart(n NUMBER,d DATE)
ON COMMIT DELETE ROWS;
```

2

```
CREATE GLOBAL TEMPORARY TABLE emp_details
ON COMMIT PRESERVE ROWS;
```

24

The definition of a global temporary table is visible to all sessions, but the data in a global temporary table is visible only to the session that inserts the data into the table.

To create a global temporary table, use the following command:

CREATE GLOBAL TEMPORARY TABLE *tablename*(

column_definition,

...)

ON COMMIT [PRESERVE | DELETE] ROWS;

The **ON COMMIT** clause indicates if the data in the table is transaction-specific (the default) or session-specific, the implications of which are:

PRESERVE ROWS: This creates a global temporary table that is session-specific. A session gets bound to the global temporary table with the first insert into the table in the session. This binding goes away at the end of the session or by issuing a TRUNCATE of the table in the session. The database truncates the table when you terminate the session.

DELETE ROWS: This creates a global temporary table that is transaction-specific. A session becomes bound to the global temporary table with a transaction's first insert into the table. The binding goes away at the end of the transaction. The database truncates the table (delete all rows) after each commit.

When you create a global temporary table in an Oracle database, you create a static table definition. Like permanent tables, global temporary tables are defined in the data dictionary. However, global temporary tables and their indexes do not automatically allocate a segment when created. Instead, temporary segments are allocated when data is first inserted. Until data is loaded in a session, the table appears empty.

Oracle Internal & Oracle Academy Use Only

Creating a Private Temporary Table

1

```
CREATE PRIVATE TEMPORARY TABLE ORA$PTT_sales_trans
    (time_id      DATE,
     amt_sold  NUMBER(8,2))
   ON COMMIT DROP DEFINITION;
```

2

```
CREATE PRIVATE TEMPORARY TABLE ORA$PTT_sales_sess
    (time_id      DATE,
     amt_sold  NUMBER(8,2))
   ON COMMIT PRESERVE DEFINITION;
```

25

Private temporary tables are temporary database objects that are dropped at the end of a transaction or session. Private temporary tables are stored in memory and the metadata and content of a private temporary table is visible only within the session that created it.

To create a private temporary table, use the following command:

CREATE PRIVATE TEMPORARY TABLE *tablename(*

column_defininition,

...)

ON COMMIT [DROP | PRESERVE] DEFINITION;

The **ON COMMIT** clause indicates if the data in the table is transaction-specific (the default) or session-specific, the implications of which are as follows:

DROP DEFINITION: This creates a private temporary table that is transaction-specific. All data in the table is lost, and the table is dropped at the end of transaction.

PRESERVE DEFINITION: This creates a private temporary table that is session-specific. All data in the table is lost, and the table is dropped at the end of the session that created the table.

Note: Names of private temporary tables must be prefixed according to the initialization parameter PRIVATE_TEMP_TABLE_PREFIX.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Managing constraints:
  - Adding and dropping a constraint
  - Enabling and disabling a constraint
  - Deferring constraints
- Creating and using temporary tables
- Creating and using external tables

26

In this section, you learn to create and use external tables.

Oracle Internal & Oracle Academy Use Only

External Tables

27

In an external table, you store the metadata in the database and the actual data outside the database. This external table can be thought of as a view that is used for running any SQL query against external data without requiring that the external data first be loaded into the database. The external table data can be queried and joined directly and in parallel without requiring that the external data first be loaded in the database.

You can use SQL, PL/SQL, and Java to query the data in an external table.

The main difference between external tables and regular tables is that externally organized tables are read-only. No data manipulation language (DML) operations are possible, and no indexes can be created on them. However, you can create an external table, and thus unload data, by using the `CREATE TABLE AS SELECT` command.

The Oracle Server provides two major access drivers for external tables:

- The `ORACLE_LOADER` access driver is used for reading data from external files whose format can be interpreted by the SQL*Loader utility. Note that not all SQL*Loader functionality is supported with external tables.
- The `ORACLE_DATAPUMP` access driver can be used to both import and export data by using a platform-independent format. The rows from a `SELECT` statement to be loaded into an external table are written as part of a `CREATE TABLE ...ORGANIZATION EXTERNAL...AS SELECT` statement by the access driver. You can then use `SELECT` to read data out of that data file. You can also create an external table definition on another system and use that data file. This allows data to be moved between Oracle databases.

Oracle Internal & Oracle Academy Use Only

Creating a Directory for the External Table

Create a DIRECTORY object that corresponds to the directory on the file system where the external data source resides.

```
CREATE OR REPLACE DIRECTORY emp_dir
AS '/…/emp_dir';

GRANT READ ON DIRECTORY emp_dir TO ora_21;
```

28

Use the CREATE DIRECTORY statement to create a directory object. A directory object specifies an alias for a directory on the server's file system where it resides. You can use directory names when referring to an external data source, rather than hard-code the operating system path name, for greater file management flexibility.

You must have CREATE ANY DIRECTORY system privileges to create directories. When you create a directory, you are automatically granted the READ and WRITE object privileges and can grant READ and WRITE privileges to other users and roles. The DBA can also grant these privileges to other users and roles.

A user needs READ privileges for all directories used in external tables for access and WRITE privileges for the log, bad, and discard file locations being used.

In addition, a WRITE privilege is necessary when the external table framework is being used to unload data.

Oracle also provides the ORACLE_DATAPUMP type, with which you can unload data (that is, read data from a table in the database and insert it into an external table) and then reload it into an Oracle database. This is a one-time operation that can be done when the table is created. After the creation and initial population is done, you cannot update, insert, or delete any rows.

Oracle Internal & Oracle Academy Use Only

Syntax

```
CREATE [OR REPLACE] DIRECTORY directory AS 'path_name';
```

In the syntax:

`OR REPLACE` Specify `OR REPLACE` to re-create the directory database object if it already exists. You can use this clause to change the definition of an existing directory without dropping, re-creating, and regranting database object privileges previously granted on the directory. Users who were previously granted privileges on a redefined directory can continue to access the directory without requiring that the privileges be regranted.

`directory` Specify the name of the directory object to be created. The maximum length of the directory name is 30 bytes. You cannot qualify a directory object with a schema name.

`'path_name'` Specify the full path name of the operating system directory to be accessed. The path name is case-sensitive.

Oracle Internal & Oracle Academy Use Only

Creating an External Table

Syntax:

```
CREATE TABLE <table_name>
        ( <col_name> <datatype>, … )
ORGANIZATION EXTERNAL
    (TYPE <access_driver_type>
     DEFAULT DIRECTORY <directory_name>
     ACCESS PARAMETERS
      (… ) )
      LOCATION ('<location_specifier>')
     REJECT LIMIT [0 | <number> | UNLIMITED];
```

30

You create external tables by using the `ORGANIZATION EXTERNAL` clause of the `CREATE TABLE` statement.

You are not, in fact, creating a table. Rather, you are creating metadata in the data dictionary that you can use to access external data. You use the `ORGANIZATION` clause to specify the order in which the data rows of the table are stored. By specifying `EXTERNAL` in the `ORGANIZATION` clause, you indicate that the table is a read-only table located outside the database. Note that the external files must already exist outside the database.

`TYPE` *`<access_driver_type>`* indicates the access driver of the external table. The access driver is the application programming interface (API) that interprets the external data for the database. If you do not specify `TYPE`, Oracle uses the default access driver, `ORACLE_LOADER`. The other option is `ORACLE_DATAPUMP`.

You use the `DEFAULT DIRECTORY` clause to specify one or more Oracle database directory objects that correspond to directories on the file system where the external data sources may reside.

The optional `ACCESS PARAMETERS` clause enables you to assign values to the parameters of the specific access driver for this external table.

Use the `LOCATION` clause to specify one external locator for each external data source. Usually, *`<location_specifier>`* is a file, but it need not be.

The `REJECT LIMIT` clause enables you to specify how many conversion errors can occur during a query of the external data before an Oracle error is returned and the query is aborted. The default value is `0`.

Oracle Internal & Oracle Academy Use Only

The syntax for using the `ORACLE_DATAPUMP` access driver is as follows:

```
CREATE TABLE extract_emps

ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP
                       DEFAULT DIRECTORY …
                       ACCESS PARAMETERS (… )
                       LOCATION (…)

                       PARALLEL 4

                       REJECT LIMIT UNLIMITED
AS

SELECT * FROM …;
```

Oracle Internal & Oracle Academy Use Only

Creating an External Table by Using ORACLE_LOADER

```
CREATE TABLE oldemp (fname char(25), lname CHAR(25))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
 DEFAULT DIRECTORY emp_dir
 ACCESS PARAMETERS
 (RECORDS DELIMITED BY NEWLINE
  FIELDS(fname POSITION ( 1:20) CHAR,
  lname POSITION (22:41) CHAR))
 LOCATION ('emp.dat'));
```

```
Table OLDEMP created.
```

32

Assume that there is a flat file that has records in the following format:

```
10,jones,11-Dec-1934
20,smith,12-Jun-1972
```

Records are delimited by new lines. The file is located at `/home/oracle/labs/sql2/emp_dir/emp.dat`.

To convert this file as the data source for an external table, whose metadata will reside in the database, you must perform the following steps:

1. Create a directory object, `emp_dir`, as follows:
   `CREATE DIRECTORY emp_dir AS '/home/oracle/labs/sql2/emp_dir' ;`
2. Run the `CREATE TABLE` command shown in the slide.

The example in the slide illustrates the table specification to create an external table for the file:

```
/home/oracle/labs/sql2/emp_dir/emp.dat
```

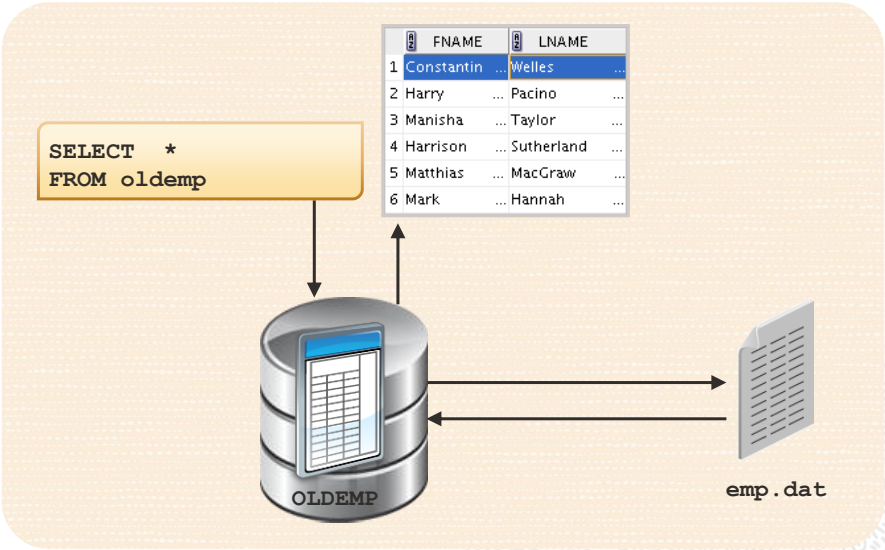
In the example, the `TYPE` specification is given only to illustrate its use. `ORACLE_LOADER` is the default access driver if not specified. The `ACCESS PARAMETERS` option provides values to parameters of the specific access driver, which are interpreted by the access driver, not by the Oracle Server.

After the `CREATE TABLE` command executes successfully, the `OLDEMP` external table can be described and queried in the same way as a relational table.

Note: The `emp_dir` directory on the file system must have write permission for user and others for the external table to work successfully.

Oracle Internal & Oracle Academy Use Only

Querying External Tables



33

An external table does not tell you how data is stored in the database. It also does not tell you how data is stored in the external source. Instead, it tells you how the external table layer must present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the data file so that it matches the external table definition.

When the database server accesses data in an external source, it calls the appropriate access driver to get the data from an external source in a form that the database server expects.

Remember that the description of the data in the data source is separate from the definition of the external table. The source file can contain more or fewer fields than there are columns in the table. Also, the data types for fields in the data source can be different from the columns in the table. The access driver takes care of ensuring that the data from the data source is processed so that it matches the definition of the external table.

Oracle Internal & Oracle Academy Use Only

Creating an External Table by Using ORACLE_DATAPUMP: Example

```
CREATE TABLE emp_ext
  (employee_id, first_name, last_name)
   ORGANIZATION EXTERNAL
    (
     TYPE ORACLE_DATAPUMP
     DEFAULT DIRECTORY emp_dir
     LOCATION
      ('emp1.exp','emp2.exp')
    )
   PARALLEL
AS
SELECT employee_id, first_name, last_name
FROM   employees;
```

```
Table EMP_EXT created.
```

34

You can perform the unload and reload operations with external tables by using the ORACLE_DATAPUMP access driver.

Note: In the context of external tables, loading data refers to the act of data being read from an external table and loaded into a table in the database. Unloading data refers to the act of reading data from a table and inserting it into an external table.

The example in the slide illustrates the table specification to create an external table by using the ORACLE_DATAPUMP access driver. Data is then populated into the two files: emp1.exp and emp2.exp.

To populate data read from the EMPLOYEES table into an external table, you must perform the following steps:

1. Create a directory object, emp_dir, as follows:
   CREATE OR REPLACE DIRECTORY emp_dir AS '/stage/Labs/sql2/emp_dir';
2. Run the CREATE TABLE command shown in the slide.

Note: The emp_dir directory is the same as created in the previous example of using ORACLE_LOADER.

You can query the external table by executing the following code:

```
SELECT * FROM emp_ext;
```

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Manage constraints
- Create and use temporary tables
- Create and use external tables

35

In this lesson, you learned how to perform the following tasks for schema object management:

- Alter tables to add or modify columns or constraints.
- Use the `ORGANIZATION EXTERNAL` clause of the `CREATE TABLE` statement to create an external table. An external table is a read-only table whose metadata is stored in the database but whose data is stored outside the database.
- Use external tables to query data without first loading it into the database.

Oracle Internal & Oracle Academy Use Only

Practice 15: Overview

This practice covers the following topics:

- Adding and dropping constraints
- Deferring constraints
- Creating external tables

36

In this practice, you use the ALTER TABLE command to add, drop, and defer constraints. You create external tables.

Oracle Internal & Oracle Academy Use Only

ORACLE

16

Retrieving Data by Using Subqueries

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** — You are here!
  - Lesson 17: Manipulating Data by Using Subqueries
- Unit 6: User Access
- Unit 7: Advanced Queries

2

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Write a multiple-column subquery
- Use scalar subqueries in SQL
- Solve problems with correlated subqueries
- Use the `EXISTS` and `NOT EXISTS` operators
- Use the `WITH` clause

3

In this lesson, you learn how to write multiple-column subqueries and subqueries in the `FROM` clause of a `SELECT` statement. You also learn to solve problems by using scalar, correlated subqueries and to use the `WITH` clause.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Retrieving data by using a subquery as a source
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

4

In this section, you learn how to retrieve data by using subquery as a source.

Oracle Internal & Oracle Academy Use Only

Retrieving Data by Using a Subquery as a Source

```
SELECT department_name, city
FROM   departments
NATURAL JOIN (SELECT l.location_id, l.city, l.country_id
              FROM   locations l
              JOIN   countries c
              ON(l.country_id = c.country_id)
              JOIN regions
              USING(region_id)
              WHERE region_name = 'Europe');
```

	DEPARTMENT_NAME	CITY
1	Human Resources	London
2	Sales	Oxford
3	Public Relations	Munich

5

You can use a subquery in the `FROM` clause of a `SELECT` statement, which is very similar to how views are used. This form of a subquery is also called an *inline* view.

A subquery in the `FROM` clause of a `SELECT` statement defines a data source for that particular `SELECT` statement only. Just like a database view, the `SELECT` statement in the subquery can be as simple or as complex as you like.

When a database view is created, the associated `SELECT` statement is stored in the data dictionary. In situations where you do not have the necessary privileges to create database views, or when you would like to test the suitability of a `SELECT` statement to become a view, you can use an inline view.

With inline views, you can have all the code needed to support the query in one place. This means that you can avoid the complexity of creating a separate database view.

The example in the slide shows how to use an inline view to display the department name and the city in Europe. The subquery in the `FROM` clause fetches the location ID, city name, and the country by joining three different tables. The output of the inner query is considered as a table for the outer query. The inner query is similar to that of a database view but does not have any physical name.

Oracle Internal & Oracle Academy Use Only

You can display the same output as in the example in the slide by performing the following two steps:

1. Create a database view:

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT l.location_id, l.city, l.country_id
FROM   locations l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe';
```

2. Join the `EUROPEAN_CITIES` view with the `DEPARTMENTS` table:

```
SELECT department_name, city
FROM   departments
NATURAL JOIN european_cities;
```

Note: You learned how to create database views in the lesson titled "Creating Views."

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

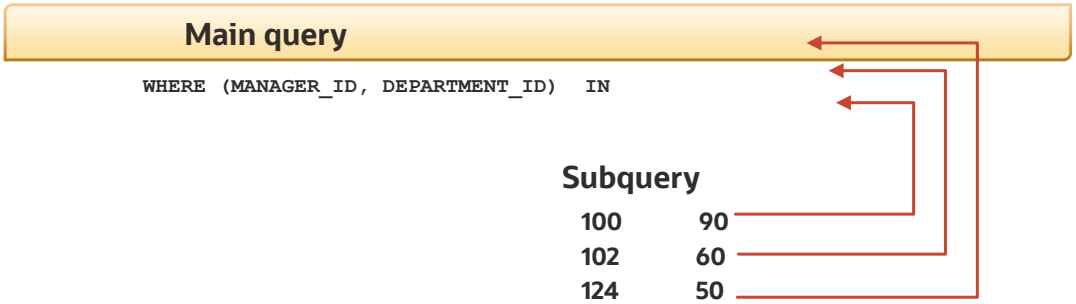
- Retrieving data by using a subquery as a source
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

7

In this section, you learn how to write a multiple-column subquery.

Oracle Internal & Oracle Academy Use Only

Multiple-Column Subqueries



Each row of the main query is compared to values from a multiple-row and multiple-column subquery.

8

So far, you have written single-row subqueries and multiple-row subqueries where only one column is returned by the inner `SELECT` statement, and this is used to evaluate the expression in the parent `SELECT` statement.

If you want to compare two or more columns, you must write a compound `WHERE` clause by using logical operators. Using multiple-column subqueries, you can combine duplicate `WHERE` conditions into a single `WHERE` clause.

Syntax

```
SELECT     column, column, ...
FROM       table
WHERE      (column, column, ...) IN
                        (SELECT column, column, ...
                                FROM   table
                                WHERE  condition);
```

The graphic in the slide illustrates that the values of `MANAGER_ID` and `DEPARTMENT_ID` from the main query are being compared with the `MANAGER_ID` and `DEPARTMENT_ID` values retrieved by the subquery. Because the number of columns that are being compared is more than one, the example qualifies as a multiple-column subquery.

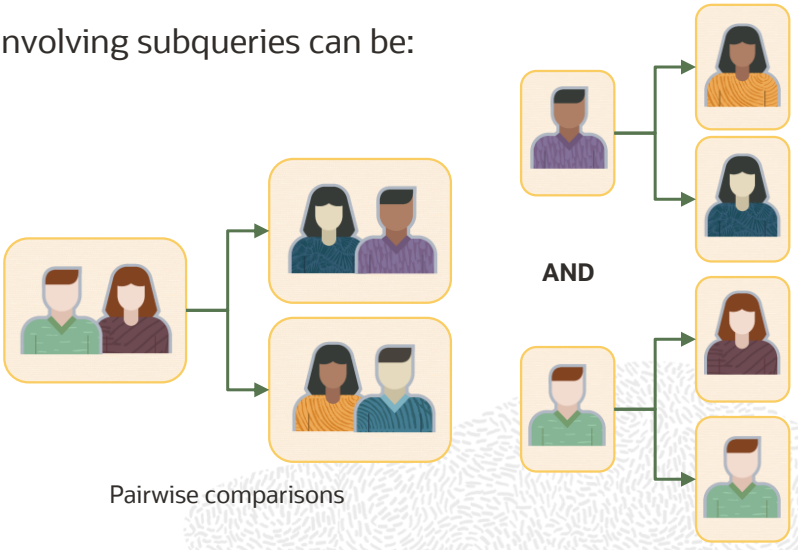
Note: Before you run the examples in the next few slides, you need to create the `empl_demo` table and populate data into it by using the `lab_06_insert_empdata.sql` file.

Oracle Internal & Oracle Academy Use Only

Column Comparisons

Multiple-column comparisons involving subqueries can be:

- Pairwise comparisons
- Nonpairwise comparisons



9

Pairwise Versus Nonpairwise Comparisons

Multiple-column comparisons involving subqueries can be nonpairwise comparisons or pairwise comparisons. If you consider the example "Display the details of the employees who work in the same department, and have the same manager, as 'Daniel'?," then you get the correct result with the following statement:

```
SELECT first_name, last_name, manager_id, department_id
FROM empl_demo
WHERE manager_id IN (SELECT manager_id
                     FROM empl_demo
                     WHERE first_name = 'Daniel')
AND department_id IN (SELECT department_id
                      FROM empl_demo
                      WHERE first_name = 'Daniel');
```

There is only one "Daniel" in the `EMPL_DEMO` table (Daniel Faviet, who is managed by employee 108 and works in department 100). However, if the subqueries return more than one row, the result might not be correct. For example, if you run the same query but substitute "John" for "Daniel," you get an incorrect result. This is because the combination of `department_id` and `manager_id` is important. To get the correct result for this query, you need a pairwise comparison.

Oracle Internal & Oracle Academy Use Only

Pairwise Comparison Subquery

Display the details of the employees who are managed by the same manager and work in the same department as the employees with EMPLOYEE_ID 199 or 174.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE (manager_id, department_id) IN
                            (SELECT manager_id, department_id
                             FROM employees
                             WHERE employee_id IN (174, 199))
AND employee_id NOT IN (174,199);
```

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	141	124	50
2	142	124	50
3	143	124	50
4	144	124	50

...

10

The example shows a pairwise comparison of the columns. It compares the values in the MANAGER_ID column and the DEPARTMENT_ID column of each row in the EMPLOYEES table with the values in the MANAGER_ID column and the DEPARTMENT_ID column for the employees with the EMPLOYEE_ID 199 or 174.

- First, the subquery to retrieve the MANAGER_ID and DEPARTMENT_ID values for the employees with the EMPLOYEE_ID 199 or 174 is executed.
- These values are compared with the MANAGER_ID column and the DEPARTMENT_ID column of each row in the EMPLOYEES table. If the values match, the row is displayed.
- In the output, the records of the employees with the EMPLOYEE_ID 199 or 174 will not be displayed. The output of the query is shown in the slide.

Oracle Internal & Oracle Academy Use Only

Nonpairwise Comparison Subquery

Display the details of the employees who are managed by the same manager as the employees with EMPLOYEE_ID 174 or 141 and work in the same department as the employees with EMPLOYEE_ID 174 or 141.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE manager_id IN
                  (SELECT manager_id
                   FROM employees
                   WHERE employee_id IN (174,141))
AND department_id IN
                  (SELECT department_id
                   FROM employees
                   WHERE employee_id IN (174,141))
AND employee_id NOT IN(174,141);
```

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	142	124	50
2	143	124	50

...

11

The example in the slide shows a nonpairwise comparison of the columns. It displays the EMPLOYEE_ID, MANAGER_ID, and DEPARTMENT_ID of an employee whose manager ID matches with any of the manager IDs of employees whose employee IDs are either 174 or 141 and DEPARTMENT_ID match any of the department IDs of employees whose employee IDs are either 174 or 141.

- First, the subquery to retrieve the MANAGER_ID values for the employees with the EMPLOYEE_ID 174 or 141 is executed.
- Similarly, the second subquery to retrieve the DEPARTMENT_ID values for the employees with the EMPLOYEE_ID 174 or 141 is executed.
- The retrieved values of the MANAGER_ID and DEPARTMENT_ID columns are compared with the MANAGER_ID and DEPARTMENT_ID column for each row in the EMPLOYEES table.
- If the MANAGER_ID column of the row in the EMPLOYEES table matches with any of the values of the MANAGER_ID retrieved by the inner subquery and if the DEPARTMENT_ID column of the row in the EMPLOYEES table matches with any of the values of the DEPARTMENT_ID retrieved by the second subquery, the record is displayed.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Retrieving data by using a subquery as a source
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

12

This section discusses the use of scalar subqueries in SQL.

Oracle Internal & Oracle Academy Use Only

Scalar Subquery Expressions

- A scalar subquery is a subquery that returns exactly one column value from one row.
- Scalar subqueries can be used in:
  - The condition and expression part of `DECODE` and `CASE`
  - All clauses of `SELECT` except `GROUP BY`
  - The `SET` clause and `WHERE` clause of an `UPDATE` statement

13

A subquery that returns exactly one column value from one row is referred to as a scalar subquery. Multiple-column subqueries that are written to compare two or more columns, using a compound `WHERE` clause and logical operators, do not qualify as scalar subqueries.

The value of the scalar subquery expression is the value of the select list item of the subquery. If the subquery returns 0 rows, the value of the scalar subquery expression is `NULL`. If the subquery returns more than one row, the Oracle Server returns an error. The Oracle Server has always supported the usage of a scalar subquery in a `SELECT` statement.

You can use scalar subqueries in:

- The condition and expression part of `DECODE` and `CASE`
- All clauses of `SELECT` except `GROUP BY`
- The `SET` clause and `WHERE` clause of an `UPDATE` statement

However, scalar subqueries are not valid expressions in the following places:

- In the `RETURNING` clause of data manipulation language (DML) statements
- As the basis of a function-based index
- In `GROUP BY` clauses and `CHECK` constraints
- In `CONNECT BY` clauses
- In statements that are unrelated to queries, such as `CREATE PROFILE`

Oracle Internal & Oracle Academy Use Only

Scalar Subqueries: Examples

- Scalar subqueries in CASE expressions:

```
SELECT employee_id, last_name,
       (CASE
        WHEN department_id =
                (SELECT department_id
                 FROM departments
                 WHERE location_id = 1800)
         THEN 'Canada' ELSE 'USA' END) location
FROM   employees;
```

20

Scalar subqueries in the SELECT statement:

```
select department_id, department_name,
      (select count(*)
       from   employees e
       where  e.department_id = d.department_id) as emp_count
from   departments d;
```

14

- The first example in the slide demonstrates that scalar subqueries can be used in CASE expressions.
- The inner query returns the value 20, which is the department ID of the department whose location ID is 1800.
- The CASE expression in the outer query uses the result of the inner query to display the employee ID, last names, and a value of Canada or USA, depending on whether the department ID of the record retrieved by the outer query is 20.
- The second example in the slide demonstrates that scalar subqueries can be used in SELECT statements.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Retrieving data by using a subquery as a source
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

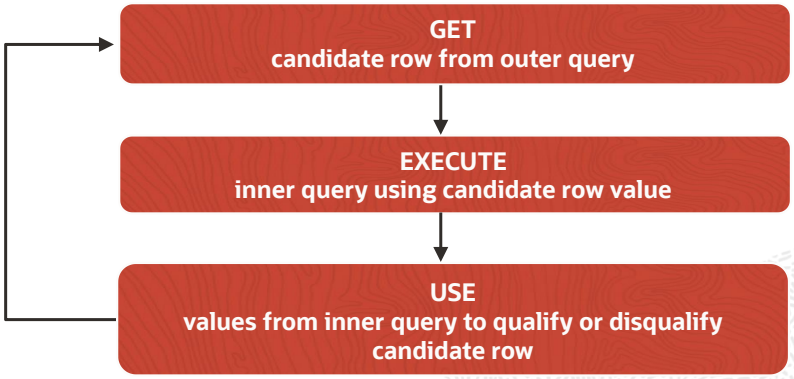
15

This section discusses how to solve problems with correlated subqueries.

Oracle Internal & Oracle Academy Use Only

Correlated Subqueries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



16

The Oracle Server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement. Whenever the parent statement is processed for a row, the correlated subquery is evaluated once for that row. The parent statement can be a `SELECT`, an `UPDATE`, or a `DELETE` statement.

Nested Subqueries Versus Correlated Subqueries

- With a normal **nested subquery**, the inner `SELECT` query runs first and executes once, returning values to be used by the main query.
- A **correlated subquery**, however, executes once for each candidate row considered by the outer query. That is, the inner query is driven by the outer query.

Nested Subquery Execution

- The inner query executes first and finds a value.
- The outer query executes once, using the value from the inner query.

Correlated Subquery Execution

- Get a candidate row (fetched by the outer query).
- Execute the inner query by using the value of the candidate row.
- Use the values resulting from the inner query to qualify or disqualify the candidate.
- Repeat until no candidate row remains.

Oracle Internal & Oracle Academy Use Only

Correlated Subqueries

The subquery references a column from a table in the parent query.

Parent Query

```
SELECT column1, column2, ...
FROM   table1 Outer_table
WHERE  column1 operator
                (SELECT  column1, column2
                      FROM    table2
                      WHERE   expr1 =
    Outer_table.expr2);
```

Subquery

17

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. That is, you use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

The Oracle Server performs a correlated subquery when the subquery references a column from a table in the parent query.

Note: You can use the `ANY` and `ALL` operators in a correlated subquery.

Oracle Internal & Oracle Academy Use Only

Using Correlated Subqueries: Example 1

Find all employees who earn more than the average salary in their department.

```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
       (SELECT AVG(salary)
            FROM   employees inner_table
            WHERE inner_table.department_id =
            outer_table.department_id);
```

	LAST_NAME	SALARY	DEPARTMENT_ID
1	King	24000	90
2	Hunold	9000	60
3	Ernst	6000	60
4	Greenberg	12008	100
5	Faviet	9000	100
6	Raphaely	11000	30

....

Each time a row from the outer query is processed, the inner query is evaluated.

18

The example in the slide finds employees who earn more than the average salary of their department. In this case, the correlated subquery specifically computes the average salary for each department.

Because both the outer query and inner query use the `EMPLOYEES` table in the `FROM` clause, an alias is given to `EMPLOYEES` in the outer `SELECT` statement for clarity. The alias makes the entire `SELECT` statement more readable. Without the alias, the query would not work properly because the inner statement would not be able to distinguish the inner table column from the outer table column.

The correlated subquery performs the following steps for each row of the `EMPLOYEES` table:

1. The `department_id` of the row is determined.
2. The `department_id` is then used to evaluate the subquery.
3. If the salary in that row is greater than the average salary in that department, then the row is returned.

The subquery is evaluated once for each row of the `EMPLOYEES` table.

Oracle Internal & Oracle Academy Use Only

Using Correlated Subqueries: Example 2

Display the details of the highest earning employees in each department.

```
SELECT department_id, employee_id, salary
FROM EMPLOYEES e
WHERE salary =
      (SELECT  MAX(DISTINCT salary)
       FROM EMPLOYEES
       WHERE e.department_id = department_id);
```

	DEPARTMENT_ID	EMPLOYEE_ID	SALARY
1	90	100	24000
2	60	103	9000
3	100	108	12008
4	30	114	11000

....

19

The example in the slide displays the details of the highest earning employees in each department. The Oracle Server evaluates a correlated subquery as follows:

1. Selects a row from the table specified in the outer query. This will be the current candidate row.
2. Stores the value of the column referenced in the subquery from this candidate row. In the example in the slide, the column referenced in the subquery is `e.department_id`.
3. Performs the subquery with its condition referencing the value from the outer query's candidate row. In the example in the slide, the `MAX(DISTINCT salary)` group function is evaluated based on the value of the `E.DEPARTMENT_ID` column obtained in step 2.
4. Evaluates the `WHERE` clause of the outer query on the basis of results of the subquery performed in step 3. This determines whether the candidate row is selected for output. In the example, the highest salary earned by employees in a department is evaluated by the subquery and is compared with the salary of the candidate row in the `WHERE` clause of the outer query. If the condition is satisfied, the candidate row's employee record is displayed.
5. Repeats the procedure for the next candidate row of the table, and so on, until all the rows in the table have been processed.

The correlation is established by using an element from the outer query in the subquery. In this example, you compare `DEPARTMENT_ID` from the table in the subquery with `DEPARTMENT_ID` from the table in the outer query.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Retrieving data by using a subquery as a source
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

20

In this section, you learn how to use `EXISTS` and `NON EXISTS` operators.

Oracle Internal & Oracle Academy Use Only

Using the EXISTS Operator

- The `EXISTS` operator tests for existence of rows in the results set of the subquery.
- If a subquery row value is found:
  - The search does not continue in the inner query
  - The condition is flagged `TRUE`
- If a subquery row value is not found:
  - The condition is flagged `FALSE`
  - The search continues in the inner query

21

Although with nesting `SELECT` statements all logical operators are valid, you can use the `EXISTS` operator.

The `EXISTS` operator is frequently used with correlated subqueries to test whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query.

The `EXISTS` operator logic is as follows:

- If the subquery returns at least one row, the operator returns `TRUE`.
- If the value does not exist, it returns `FALSE`.

Accordingly, you can use the `NOT EXISTS` operator to test whether a value retrieved by the outer query is not a part of the results set of the values retrieved by the inner query.

Oracle Internal & Oracle Academy Use Only

Using the EXISTS Operator

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT NULL
                FROM   employees
                WHERE  manager_id =
                       outer.employee_id);
```

	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	108	Greenberg	FI_MGR	100
6	114	Raphaely	PU_MAN	30
7	120	Weiss	ST_MAN	50
8	121	Fripp	ST_MAN	50

...

22

Observe the example in the slide, which displays all the managers in the `EMPLOYEES` table using the `EXISTS` operator.

The `EXISTS` operator ensures that the search in the inner query does not continue when at least one match is found for the manager and employee number by the condition:

```
WHERE manager_id = outer.employee_id
```

Note that the inner `SELECT` query does not need to return a specific value, so a constant can be selected.

Oracle Internal & Oracle Academy Use Only

Find All Departments That Do Not Have Any Employees

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT NULL
                  FROM   employees
                  WHERE  department_id = d.department_id);
```

	DEPARTMENT_ID	DEPARTMENT_NAME
1	120	Treasury
2	130	Corporate Tax
3	140	Control And Credit
4	150	Shareholder Services
5	160	Benefits
6	170	Manufacturing
7	180	Construction
8	190	Contracting
9	200	Operations
10	210	IT Support

...

All Rows Fetched: 16

23

Using the NOT EXISTS Operator

Alternative Solution

A NOT IN construct can be used as an alternative for a NOT EXISTS operator, as shown in the following example:

```
SELECT department_id, department_name
FROM   departments
WHERE  department_id NOT IN (SELECT department_id
                             FROM   employees);
```

However, NOT IN evaluates to FALSE if any member of the set is a NULL value. Therefore, your query will not return any rows even if there are rows in the departments table that satisfy the WHERE condition.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Retrieving data by using a subquery as a source
- Writing a multiple-column subquery
- Using scalar subqueries in SQL
- Solving problems with correlated subqueries
- Using the `EXISTS` and `NOT EXISTS` operators
- Using the `WITH` clause

24

This section discusses the use of the `WITH` clause.

Oracle Internal & Oracle Academy Use Only

WITH Clause

- Using the `WITH` clause, you can use the same query block in a `SELECT` statement when it occurs more than once within a complex query.
- The `WITH` clause retrieves the results of a query block and stores them in the user's temporary tablespace.
- The `WITH` clause can improve performance.

25

Using the `WITH` clause, you can define a query block before using it in a query.

The `WITH` clause (formally known as `subquery_factoring_clause`) enables you to reuse the same query block in a `SELECT` statement when it occurs more than once within a complex query. This is particularly useful when a query has many references to the same query block and there are joins and aggregations. This also helps in reducing the cost to evaluate the query block multiple times.

Using the `WITH` clause, the Oracle Server retrieves the results of a query block and stores them in the user's temporary tablespace. This can improve performance.

`WITH` Clause Benefits

- Makes the query easy to read
- Evaluates a clause only once, even if it appears multiple times in the query
- In most cases, may improve performance for large queries

Oracle Internal & Oracle Academy Use Only

WITH Clause: Example

```
WITH CNT_DEPT AS
(
SELECT department_id,
 COUNT(*) NUM_EMP
FROM EMPLOYEES
GROUP BY department_id
)
SELECT employee_id,
 SALARY/NUM_EMP
FROM EMPLOYEES E
JOIN CNT_DEPT C
ON (e.department_id = c.department_id);
```

	EMPLOYEE_ID	SALARY/NUM_EMP
1	100	8000
2	103	1800
3	108	2001.3333333333333333333333333333333333
4	110	1366.6666666666666666666666666666666667
5	114	1833.3333333333333333333333333333333333
6	137	80
7	139	60

...

26

The SQL code in the slide is an example of a situation in which you can improve performance and write SQL more simply by using the WITH clause.

The query creates the query name as CNT_DEPT and then uses it in the body of the main query. Here, you perform a math operation by dividing the salary of an employee with the total number of employees in each department.

Internally, the WITH clause is resolved either as an inline view or a temporary table. The optimizer chooses the appropriate resolution depending on the cost or benefit of temporarily storing the results of the WITH clause.

WITH Clause Usage Notes

- You can use it only with SELECT statements.
- A query name is visible to all WITH element query blocks (including their subquery blocks) defined after it and the main query block itself (including its subquery blocks).
- When the query name is the same as an existing table name, the parser searches from the inside out, and the query block name takes precedence over the table name.
- The WITH clause can hold more than one query. Each query is then separated by a comma.

Oracle Internal & Oracle Academy Use Only

Recursive WITH Clause

The Recursive `WITH` clause:

- Enables formulation of recursive queries
- Creates a query with a name, called the Recursive `WITH` element name
- Contains two types of query block members: an anchor and a recursive
- Is `ANSI` compatible

27

Using the extension of the `WITH` clause, you can also formulate recursive queries.

- Recursive `WITH` defines a recursive query with a name, the Recursive `WITH` element name.
- The Recursive `WITH` element definition must contain at least two query blocks: an anchor member and a recursive member. There can be multiple anchor members, but there can be only a single recursive member.
- The anchor member must appear before the recursive member, and it cannot reference *query_name*. The anchor member can be composed of one or more query blocks combined by the set operators, for example, `UNION ALL`, `UNION`, `INTERSECT`, or `MINUS`.
- The recursive member must follow the anchor member and must reference *query_name* exactly once. You must combine the recursive member with the anchor member by using the `UNION ALL` set operator.
- The Recursive `WITH` clause complies with the American National Standards Institute (ANSI) standard.
- Recursive `WITH` can be used to query hierarchical data, such as organization charts.

Oracle Internal & Oracle Academy Use Only

Recursive WITH Clause: Example

FLIGHTS Table

	SOURCE	DESTIN	FLIGHT_TIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8

1

```
WITH Reachable_From (Source, Destin, TotalFlightTime) AS
(
    SELECT Source, Destin, Flight_time
    FROM Flights
  UNION ALL
      SELECT incoming.Source, outgoing.Destin,
             incoming.TotalFlightTime+outgoing.Flight_time
      FROM Reachable_From incoming, Flights outgoing
      WHERE incoming.Destin = outgoing.Source
)
SELECT Source, Destin, TotalFlightTime
FROM Reachable_From;
```

2

3

	SOURCE	DESTIN	TOTALFLIGHTTIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8
4	Los Angeles	Boston	6.9
5	San Jose	New York	7.1
6	San Jose	Boston	8.2

28

Example 1 in the slide displays records from a `FLIGHTS` table describing flights between two cities.

Using the query in example 2, you query the `FLIGHTS` table to display the total flight time between any source and destination.

The `WITH` clause in the query, which is named `Reachable_From`, has a `UNION ALL` query with two branches:

- The first branch is the *anchor* branch, which selects all the rows from the `Flights` table.
- The second branch is the recursive branch. It joins the contents of `Reachable_From` to the `Flights` table to find other cities that can be reached and adds these to the content of `Reachable_From`. The operation will finish when no more rows are found by the recursive branch.

Example 3 displays the result of the query that selects everything from the `WITH` clause element `Reachable_From`.

For details, see:

- *Oracle Database SQL Language Reference 19c.*
- *Oracle Database Data Warehousing Guide 19c.*

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Write a multiple-column subquery
- Use scalar subqueries in SQL
- Solve problems with correlated subqueries
- Use the `EXISTS` and `NOT EXISTS` operators
- Use the `WITH` clause

29

You can use multiple-column subqueries to combine multiple `WHERE` conditions in a single `WHERE` clause. Column comparisons in a multiple-column subquery can be pairwise comparisons or nonpairwise comparisons.

You can use a subquery to define a table to be operated on by a containing query.

Scalar subqueries can be used in:

- The condition and expression part of `DECODE` and `CASE`
- All clauses of `SELECT` except `GROUP BY`
- A `SET` clause and `WHERE` clause of the `UPDATE` statement

The Oracle Server performs a correlated subquery when the subquery references a column from a table referred to in the parent statement. A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a `SELECT` statement. Using the `WITH` clause, you can reuse the same query when it is costly to re-evaluate the query block and it occurs more than once within a complex query.

Oracle Internal & Oracle Academy Use Only

Practice 16: Overview

This practice covers the following topics:

- Creating multiple-column subqueries
- Writing correlated subqueries
- Using the `EXISTS` operator
- Using scalar subqueries
- Using the `WITH` clause

30

In this practice, you write multiple-column subqueries and correlated and scalar subqueries. You also solve problems by writing the `WITH` clause.

Oracle Internal & Oracle Academy Use Only

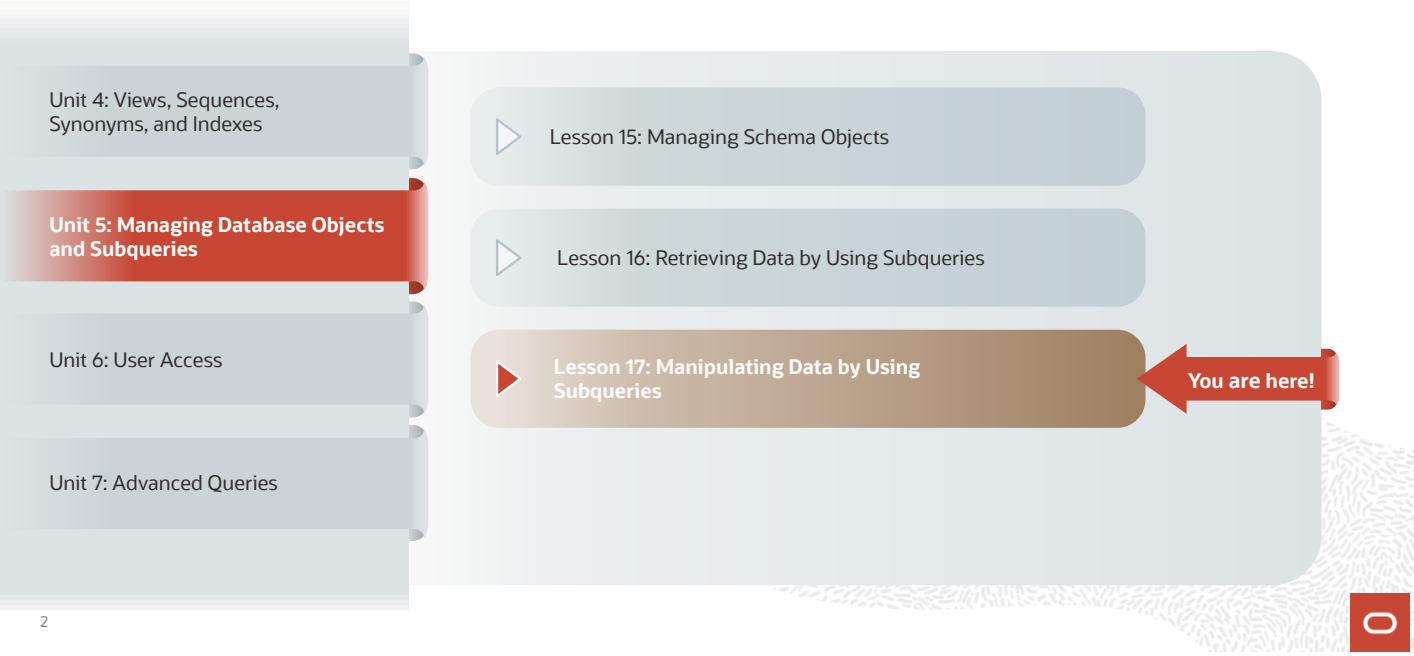
ORACLE

17

Manipulating Data by Using Subqueries

Oracle Internal & Oracle Academy Use Only

Course Roadmap



In Unit 5, you will learn to use SQL statements to query and display data from multiple tables using Joins, 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:

- Use subqueries to manipulate data
- Insert values by using a subquery as a target
- Use the `WITH CHECK OPTION` keyword on DML statements
- Use correlated subqueries to update and delete rows

3

In this lesson, you learn how to manipulate data in the Oracle database by using subqueries. You also learn how to solve problems by using correlated subqueries.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

4

In this section, you learn how to use subqueries to manipulate data.

Oracle Internal & Oracle Academy Use Only

Using Subqueries to Manipulate Data

You can use subqueries in data manipulation language (DML) statements to:

- Retrieve data by using an inline view
- Copy data from one table to another
- Update data in one table based on the values of another table
- Delete rows from one table based on rows in another table

5

Subqueries can be used to retrieve data from a table that you can use as input to an `INSERT` into a different table. In this way, you can easily copy large volumes of data from one table to another with one single `SELECT` statement.

Similarly, you can use subqueries to perform mass updates and deletes by using them in the `WHERE` clause of the `UPDATE` and `DELETE` statements. You can also use subqueries in the `FROM` clause of a `SELECT` statement. This is called an inline view.

Note: You learned how to update and delete rows based on another table in the course titled *Oracle Database: SQL Workshop I*.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

6

In this section, you learn how to insert values into a table by using a subquery as a target.

Oracle Internal & Oracle Academy Use Only

Inserting by Using a Subquery as a Target

```
INSERT INTO (SELECT l.location_id, l.city, l.country_id
             FROM   loc l
             JOIN   countries c
             ON(l.country_id = c.country_id)
             JOIN regions USING(region_id)
             WHERE region_name = 'Europe')
VALUES (3300, 'Cardiff', 'UK');
```

```
1 row inserted.
```

7

You can use a subquery in place of the table name in the `INTO` clause of the `INSERT` statement. The `SELECT` list of this subquery must have the same number of columns as the column list of the `VALUES` clause.

Any rules on the columns of the base table must be followed for the `INSERT` statement to work successfully. For example, you cannot put in a duplicate location `ID` or leave out a value for a mandatory `NOT NULL` column.

By using subqueries in this way, you can avoid having to create a view just for performing an `INSERT`.

The example in the slide uses a subquery in the place of `LOC` to create a record for a new European city.

Note: You can also perform the `INSERT` operation on the `EUROPEAN_CITIES` view by using the following code:

```
INSERT INTO european_cities
VALUES (3300,'Cardiff','UK');
```

In the example in the slide, the `loc` table is created by running the following statement:

```
CREATE TABLE loc AS SELECT * FROM locations;
```

Oracle Internal & Oracle Academy Use Only

Inserting by Using a Subquery as a Target

Verify the results from the INSERT statement in the previous slide.

```
SELECT location_id, city, country_id
FROM   loc;
```

...

20	2900	Geneva	CH
21	3000	Bern	CH
22	3100	Utrecht	NL
23	3200	Mexico City	MX
24	3300	Cardiff	UK

8

The example in the slide shows that the insert via the inline view created a new record in the base table LOC.

The following example shows the results of the subquery that was used to identify the table for the INSERT statement.

```
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe';
```

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

9

In this section, you learn how to use the `WITH CHECK OPTION` keyword on DML statements.

Oracle Internal & Oracle Academy Use Only

Using the WITH CHECK OPTION Keyword on DML Statements

The `WITH CHECK OPTION` keyword prohibits you from changing rows that are not in the subquery.

```
INSERT INTO ( SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
              (SELECT country_id
               FROM countries
               NATURAL JOIN regions
               WHERE region_name =  Europe')
               WITH CHECK OPTION )
VALUES (3600, 'Washington', 'US');
```

Error report:
SQL Error: ORA-01402: view `WITH CHECK OPTION` where-clause violation
01402. 00000 - "view `WITH CHECK OPTION` where-clause violation"
***Cause:**
***Action:**

10

When you specify the `WITH CHECK OPTION` keyword, it indicates that if the subquery is used in place of a table in an `INSERT`, `UPDATE`, or `DELETE` statement, changes are permitted to that table only if those changes will produce rows that are included in the subquery.

The example in the slide shows how to use an inline view with `WITH CHECK OPTION`. The `INSERT` statement prevents the creation of records in the `LOC` table for a city that is not in Europe.

The following example executes successfully because of the changes in the `VALUES` list.

```
INSERT INTO (SELECT location_id, city, country_id
             FROM   loc
             WHERE  country_id IN
             (SELECT country_id
              FROM countries
              NATURAL JOIN regions
              WHERE region_name = 'Europe')
              WITH CHECK OPTION)
VALUES (3500, 'Berlin', 'DE');
```

Oracle Internal & Oracle Academy Use Only

The use of an inline view with `WITH CHECK OPTION` provides an easy method to prevent changes to the table.

To prevent the creation of a non-European city, you can also use a database view by performing the following steps:

1. Create a database view:

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT location_id, city, country_id
FROM   locations
WHERE  country_id in
  (SELECT country_id
   FROM countries
   NATURAL JOIN regions
   WHERE region_name = 'Europe')
WITH CHECK OPTION;
```

2. Verify the results by inserting data:

```
INSERT INTO european_cities
VALUES (3400,'New York','US');
```

The second step produces the same error as shown in the slide.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

12

In this section, you learn how to use correlated subqueries to update and delete rows.

Oracle Internal & Oracle Academy Use Only

Correlated UPDATE

Use a correlated subquery to update rows in one table based on rows from another table.

Syntax:

```
UPDATE table1 alias1
SET    column = (SELECT expression
                 FROM   table2 alias2
                 WHERE  alias1.column =
                        alias2.column);
```

13

In the case of the UPDATE statement, you can use a correlated subquery to update rows in one table based on rows from another table.

Oracle Internal & Oracle Academy Use Only

Using Correlated UPDATE

- Denormalize the EMPL6 table by adding a column to store the department name.
- Populate the table by using a correlated update.

```
ALTER TABLE empl6
ADD(department_name VARCHAR2(25));
```

```
Table EMPL6 altered.
```

```
UPDATE empl6 e
SET    department_name =
             (SELECT department_name
           FROM   departments d
           WHERE  e.department_id = d.department_id);
```

```
107 rows updated.
```

14

The example in the slide denormalizes the EMPL6 table by adding a column to store the department name and then populates the table by using a correlated update.

Following is another example for a correlated update.

Problem Statement

The REWARDS table has a list of employees who have exceeded expectations in their performance. Use a correlated subquery to update rows in the EMPL6 table based on rows from the REWARDS table:

```
UPDATE empl6
SET    salary = (SELECT empl6.salary + rewards.pay_raise
                 FROM   rewards
                 WHERE  employee_id  =
                        empl6.employee_id
                 AND  payraise_date =
                      (SELECT MAX(payraise_date)
                      FROM   rewards
                      WHERE  employee_id = empl6.employee_id))
WHERE  empl6.employee_id
IN     (SELECT employee_id FROM rewards);
```

Oracle Internal & Oracle Academy Use Only

This example uses the REWARDS table. The REWARDS table has the following columns: EMPLOYEE_ID, PAY_RAISE, and PAYRAISE_DATE.

Every time an employee gets a pay raise, a record with details such as the employee ID, the amount of the pay raise, and the date of receipt of the pay raise is inserted into the REWARDS table. The REWARDS table can contain more than one record for an employee. The PAYRAISE _DATE column is used to identify the most recent pay raise received by an employee.

In the example, the SALARY column in the EMPL6 table is updated to reflect the latest pay raise received by the employee. This is done by adding the current salary of the employee with the corresponding pay raise from the REWARDS table.

Oracle Internal & Oracle Academy Use Only

Correlated DELETE

Use a correlated subquery to delete rows in one table based on rows from another table.

Syntax:

```
DELETE FROM table1 alias1
WHERE  column operator
               (SELECT expression
                FROM   table2 alias2
                WHERE  alias1.column = alias2.column);
```

16

In the case of a DELETE statement, you can use a correlated subquery to delete only those rows that also exist in another table.

For example, if you decide that you will maintain only the last four job history records in the JOB_HISTORY table, when an employee transfers to a fifth job, you delete the oldest JOB_HISTORY row by looking up the JOB_HISTORY table for MIN(START_DATE) for the employee. The following code illustrates how the preceding operation can be performed using a correlated DELETE:

```
DELETE FROM job_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE =
             (SELECT MIN(start_date)
              FROM job_history JH
              WHERE JH.employee_id = E.employee_id)
              AND 5 >  (SELECT COUNT(*)
                        FROM job_history JH
                        WHERE JH.employee_id = E.employee_id
                        GROUP BY EMPLOYEE_ID
                        HAVING COUNT(*) >= 4));
```

Oracle Internal & Oracle Academy Use Only

Using Correlated DELETE

Use a correlated subquery to delete only those rows from the EMPL6 table that also exist in the EMP_HISTORY table.

```
DELETE FROM empl6 E
WHERE employee_id =
           (SELECT employee_id
            FROM   employee_history
            WHERE  employee_id = E.employee_id);
```

17

Two tables are used in this example. They are:

- The EMPL6 table, which provides details of all the current employees
- The EMPLOYEE_HISTORY table, which provides details of previous employees

EMPLOYEE_HISTORY contains data regarding previous employees, so it would be erroneous if the same employee's record existed in both the EMPL6 and EMPLOYEE_HISTORY tables. You can delete such erroneous records by using the correlated subquery shown in the slide.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Manipulate data by using subqueries
- Insert values by using a subquery as a target
- Use the `WITH CHECK OPTION` keyword on DML statements
- Use correlated subqueries with `UPDATE` and `DELETE` statements

18

In this lesson, you should have learned how to manipulate data in the Oracle database by using subqueries. You also learned how to use the `WITH CHECK OPTION` keyword on DML statements and use correlated subqueries with `UPDATE` and `DELETE` statements.

Oracle Internal & Oracle Academy Use Only

Practice 17: Overview

This practice covers the following topics:

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

19

In this practice, you learn the concepts of manipulating data by using subqueries, using `WITH CHECK OPTION`, and using correlated subqueries to `UPDATE` and `DELETE` rows.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

18

Controlling User Access

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

You are here!

Unit 7: Advanced Queries

2

In Unit 6, you will learn how to create users, control their access to database objects, and revoke permissions from users.

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Differentiate system privileges from object privileges
- Grant privileges on tables
- Grant roles
- Distinguish between privileges and roles

3

In this lesson, you learn how to control database access to specific objects and add new users with different levels of access privileges.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

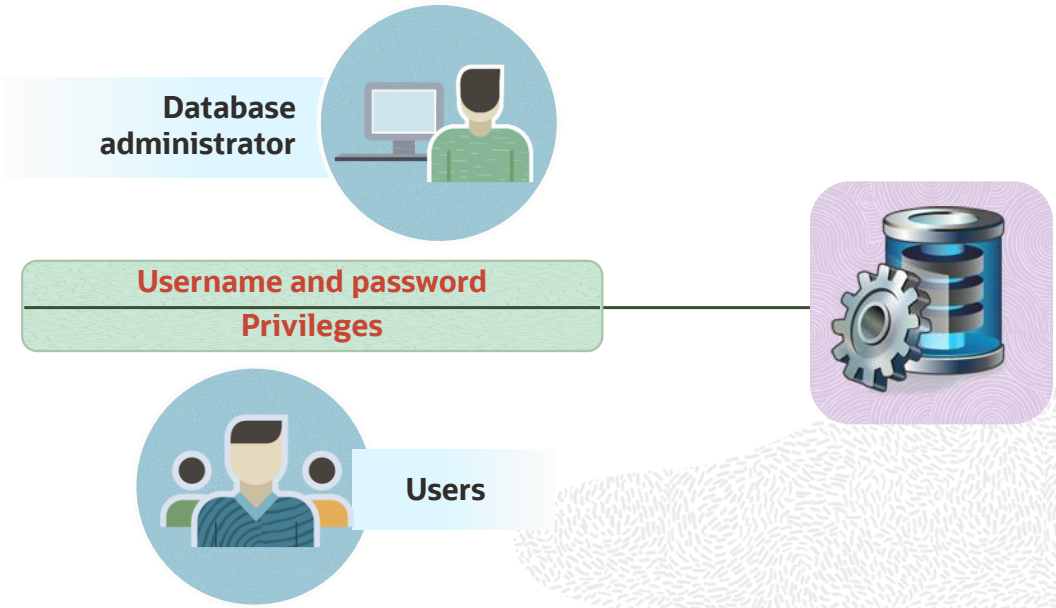
- System privileges
- Creating a role
- Object privileges
- Revoking object privileges

4

This section discusses system privileges.

Oracle Internal & Oracle Academy Use Only

Controlling User Access



5

In a multiple-user environment, you want to maintain security of database access and use.

With Oracle Server database security, you can do the following:

- Control database access
- Give access to specific objects in the database
- Confirm given and received privileges with the Oracle data dictionary

Database security can be classified into two categories:

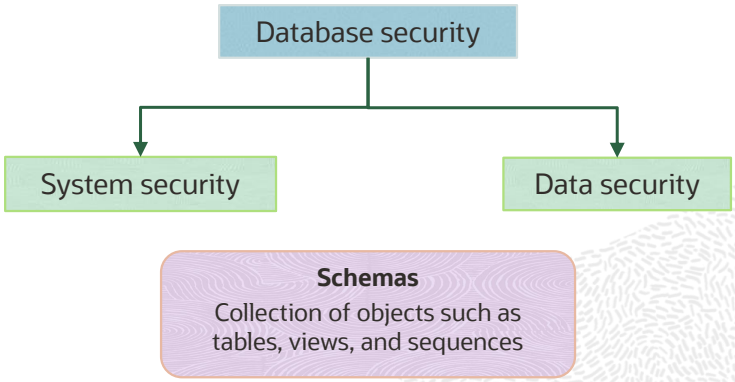
- **System security** covers access and use of the database at the system level, such as the username and password, the disk space allocated to users, and the system operations that users can perform.
- **Data security** covers access and use of the database objects and the actions that users can perform on the objects.

Oracle Internal & Oracle Academy Use Only

Privileges

There are two types of privileges:

- System privileges: Performing a particular action within the database
- Object privileges: Manipulating the content of the database objects



6

What is a **privilege**?

A privilege is the right to execute particular SQL statements.

The database administrator (DBA) is a high-level user who has the ability to create users and grant users access to the database and its objects.

You as a user will require system privileges to gain access to the database and object privileges to manipulate the content of the objects in the database. You can also be given the privilege to grant additional privileges to other users or to roles, which are named groups of related privileges.

Schemas

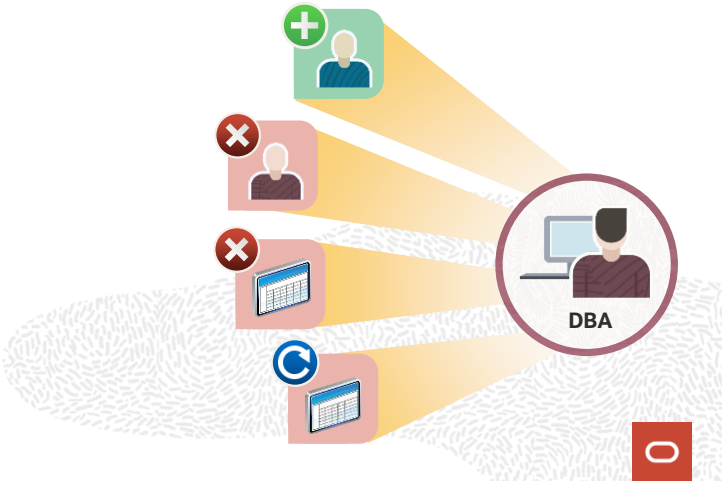
A schema is a collection of objects such as tables, views, and sequences. The schema is owned by a database user and has the same name as the user.

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type, whereas an object privilege provides the user the ability to perform a particular action on a specific schema object.

Oracle Internal & Oracle Academy Use Only

System Privileges

- More than 200 privileges are available.
- The DBA has high-level system privileges for tasks such as:
  - Creating new users
  - Removing users
  - Removing tables
  - Backing up tables



7

More than 200 distinct system privileges are available for users and roles. Typically, system privileges are provided by the DBA.

The table `SYSTEM_PRIVILEGE_MAP` contains all the system privileges available, based on the version release. This table is also used to map privilege type numbers to type names.

Typical DBA Privileges

System Privilege	Operations Authorized
`CREATE USER`	Grantee can create other Oracle users.
`DROP USER`	Grantee can drop another user.
`DROP ANY TABLE`	Grantee can drop a table in any schema.
`BACKUP ANY TABLE`	Grantee can back up any table in any schema with the export utility.
`SELECT ANY TABLE`	Grantee can query tables, views, or materialized views in any schema.
`CREATE ANY TABLE`	Grantee can create tables in any schema.

Oracle Internal & Oracle Academy Use Only

Creating Users

The DBA creates users with the `CREATE USER` statement.

```
CREATE USER user
IDENTIFIED BY   password;
```

```
CREATE USER  demo
IDENTIFIED BY demo;
```

```
User DEMO created.
```

8

The DBA creates the user by executing the `CREATE USER` statement. The user does not have any privileges at this point. The DBA can then grant privileges to that user. These privileges determine what the user can do at the database level.

The slide gives the abridged syntax for creating a user.

In the syntax:

user	Is the name of the user to be created
password	Specifies that the user must log in with this password

For more information, see the *Oracle Database SQL Language Reference* for Oracle Database19*c*.

Note: Starting with Oracle Database 11*g*, passwords are case-sensitive.

Oracle Internal & Oracle Academy Use Only

User System Privileges

- After a user is created, the DBA can grant specific system privileges to that user.

```
GRANT privilege [, privilege...]
TO user [, user| role, PUBLIC...];
```

- An application developer, for example, may have the following system privileges:
  - CREATE SESSION
  - CREATE TABLE
  - CREATE SEQUENCE
  - CREATE VIEW
  - CREATE PROCEDURE

9

Typical User Privileges

After the DBA creates a user, the DBA can assign privileges to that user.

System Privilege	Operations Authorized
CREATE SESSION	Connect to the database.
CREATE TABLE	Create tables in the user's schema.
CREATE SEQUENCE	Create a sequence in the user's schema.
CREATE VIEW	Create a view in the user's schema.
CREATE PROCEDURE	Create a stored procedure, function, or package in the user's schema.

In the syntax:

privilege Is the system privilege to be granted

user |role|PUBLIC Is the name of the user, the name of the role, or PUBLIC (which designates that every user is granted the privilege)

Note: You can find the current system privileges in the SESSION_PRIVS dictionary view. As you are already aware, data dictionary is a collection of tables and views created and maintained by the Oracle Server. They contain information about the database.

Oracle Internal & Oracle Academy Use Only

Granting System Privileges

The DBA can grant specific system privileges to a user.

```
GRANT  create session, create table,
       create sequence, create view
TO     demo;
```

```
Grant succeeded.
```

10

The DBA uses the `GRANT` statement to allocate system privileges to the user. After the user has been granted the privileges, the user can immediately use those privileges.

In the example in the slide, the `demo` user has been assigned the privileges to create sessions, tables, sequences, and views.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

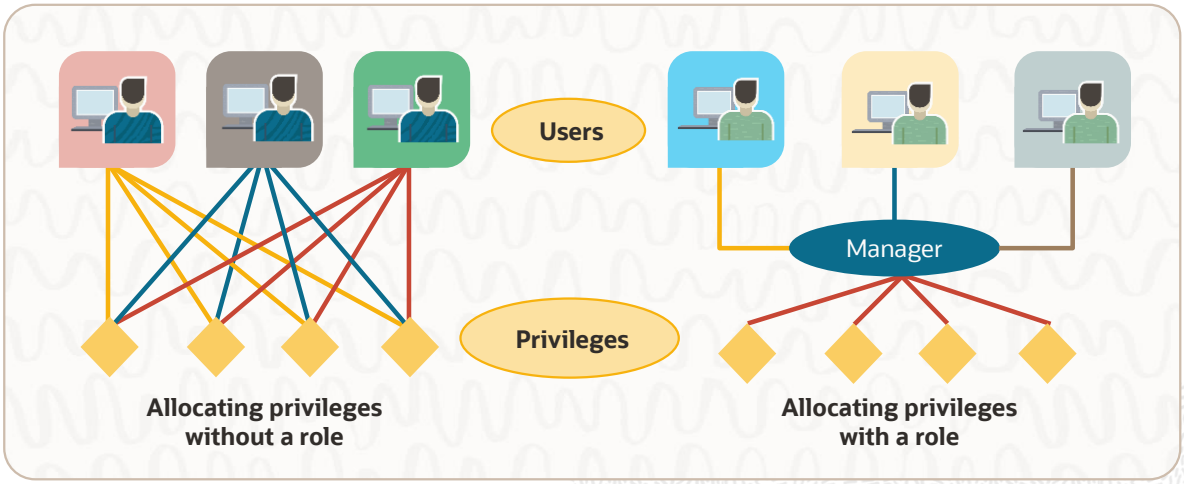
- System privileges
- Creating a role
- Object privileges
- Revoking object privileges

11

In this section, you will learn how to create a role.

Oracle Internal & Oracle Academy Use Only

What Is a Role?



12

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges.

A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application.

Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and assign the role to users.

Syntax

```
CREATE    ROLE role;
```

In the syntax:

role	Is the name of the role to be created

After the role is created, the DBA can use the `GRANT` statement to assign the role to users as well as assign privileges to the role.

Note: A role is not a schema object; therefore, any user can add privileges to a role.

Oracle Internal & Oracle Academy Use Only

Creating and Granting Privileges to a Role

- Create a role:

```
CREATE ROLE manager;
```

- Grant privileges to a role:

```
GRANT create table, create view
TO manager;
```

- Grant a role to users:

```
GRANT manager TO alice;
```

13

Creating a Role

The example in the slide creates a `manager` role. The manager is then enabled to create tables and views by granting create table and view privileges. It then grants user `alice` the role of a manager. Now `alice` can create tables and views.

If users have multiple roles granted to them, they receive all the privileges associated with all the roles.

Oracle Internal & Oracle Academy Use Only

Changing Your Password

- The DBA creates your user account and initializes your password.
- You can change your password by using the `ALTER USER` statement.

```
ALTER USER demo
IDENTIFIED BY employ;
```

14

The DBA creates an account and initializes a password for every user. You can change your password by using the `ALTER USER` statement.

The example in the slide shows that the `demo` user changes the password to `employ` by using the `ALTER USER` statement.

Syntax

```
ALTER USER user IDENTIFIED BY password;
```

In the syntax:

user	Is the name of the user
password	Specifies the new password

Although this statement can be used to change your password, there are many other options. Remember that you must have the `ALTER USER` privilege to change any other option.

For more information, see the *Oracle Database SQL Language Reference* for Oracle Database 19c.

Note: SQL*Plus has a `PASSWORD` command (`PASSW`) that can be used to change the password of a user when the user is logged in. This command is not available in Oracle SQL Developer.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- System privileges
- Creating a role
- Object privileges
- Revoking object privileges

15

This section discusses object privileges.

Oracle Internal & Oracle Academy Use Only

Object Privileges

Object Privilege	Table	View	Sequence
ALTER	✓		✓
DELETE	✓	✓	
INDEX	✓		
INSERT	✓	✓	
REFERENCES	✓		
SELECT	✓	✓	✓
UPDATE	✓	✓	

16

When an object privilege is given to you, it means that you have a privilege or right to perform a particular action on a specific table, view, sequence, or procedure.

Each object has a particular set of grantable privileges. The table in the slide lists the privileges for various objects. Note that the only privileges that apply to a sequence are `SELECT` and `ALTER`.

`UPDATE`, `REFERENCES`, and `INSERT` can be restricted by specifying a subset of updatable columns.

A `SELECT` privilege can be restricted by creating a view with a subset of columns and granting the `SELECT` privilege only on the view. A privilege granted on a synonym is converted to a privilege on the base table referenced by the synonym.

Note: With the `REFERENCES` privilege, you can ensure that other users can create `FOREIGN KEY` constraints that reference your table.

Oracle Internal & Oracle Academy Use Only

Object Privileges

- Object privileges vary from object to object.
- An owner:
  - has all the privileges on the object
  - Can give specific privileges on the objects he owns
- Syntax:

```
GRANT          object_priv [(columns)]|ALL
ON             object
TO             {user|role|PUBLIC}
[WITH GRANT OPTION];
```

17

Granting Object Privileges

You can grant different object privileges for different types of schema objects.

A user automatically has all object privileges for schema objects contained in the user's schema. A user can grant any object privilege on any schema object that the user owns to any other user or role.

If the grant includes `WITH GRANT OPTION`, the grantee can further grant the object privilege to other users; otherwise, the grantee can use the privilege but cannot grant it to other users.

In the syntax:

object_priv	Is the object privilege to be granted
`ALL`	Specifies all object privileges
columns	Specifies the column from a table or view on which privileges are granted
`ON` *object*	Is the object on which the privileges are granted
`TO`	Identifies to whom the privilege is granted
`PUBLIC`	Grants object privileges to all users
`WITH GRANT OPTION`	Enables the grantee to grant the object privileges to other users and roles

Note: In the syntax, *schema* is the same as the owner's name.

Oracle Internal & Oracle Academy Use Only

Granting Object Privileges

- Grant query privileges on the EMPLOYEES table:

```
GRANT  select
ON     employees
TO     demo;
```

- Grant privileges to update specific columns to users and roles:

```
GRANT  update (department_name, location_id)
ON     departments
TO     demo, manager;
```

18

Guidelines

- To grant privileges on an object, the object must be in your own schema, or you must have been granted the object privileges WITH GRANT OPTION.
- An object owner can grant any object privilege on the object to any other user or role of the database.
- The owner of an object automatically acquires all object privileges on that object.

The first example grants the demo user the privilege to query your EMPLOYEES table.

The second example grants UPDATE privileges on specific columns in the DEPARTMENTS table to demo and to the manager role.

For example, if your schema is oraxx, and the demo user now wants to use a SELECT statement to obtain data from your EMPLOYEES table, the syntax he or she must use is:

```
SELECT  * FROM oraxx.employees;
```

Alternatively, the demo user can create a synonym for the table and issue a SELECT statement from the synonym:

```
CREATE SYNONYM emp FOR oraxx.employees;

SELECT * FROM emp;
```

Note: DBAs generally allocate system privileges; any user who owns an object can grant object privileges.

Oracle Internal & Oracle Academy Use Only

Passing On Your Privileges

- Give a user authority to pass along privileges:

```
GRANT  select, insert
ON     departments
TO     demo
WITH GRANT OPTION;
```

- Allow all users on the system to query data from DEPARTMENTS table:

```
GRANT  select
ON      departments
TO      PUBLIC;
```

19

Oracle Internal & Oracle Academy Use Only

WITH GRANT OPTION Keyword

A privilege that is granted with the WITH GRANT OPTION clause can be passed on to other users and roles by the grantee. Object privileges granted with the WITH GRANT OPTION clause are revoked when the grantor's privilege is revoked.

You can specify WITH GRANT OPTION only when granting to a user or to PUBLIC, not when granting to a role. The grantor must meet one or more of the below criteria.

The grantor:

- Must be the object owner; otherwise, the grantor must have object access with GRANT OPTION from the user
- Must have the GRANT ANY OBJECT PRIVILEGE system privilege and an object privilege on the object

The example in the slide gives the demo user access to your DEPARTMENTS table with the privileges to query the table and add rows to the table. You can also observe that demo can give others these privileges.

PUBLIC Keyword

An owner of a table can grant access to all users by using the PUBLIC keyword. The second example allows all users on the system to query data from the DEPARTMENTS table.

Confirming Granted Privileges

Data Dictionary View	Description
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
USER_ROLE_PRIVS	Roles accessible by the user
USER_SYS_PRIVS	System privileges granted to the user
USER_TAB_PRIVS_MADE	Object privileges granted on the user's objects
USER_TAB_PRIVS_RECD	Object privileges granted to the user
USER_COL_PRIVS_MADE	Object privileges granted on the columns of the user's objects
USER_COL_PRIVS_RECD	Object privileges granted to the user on specific columns

20

If you attempt to perform an unauthorized operation, such as deleting a row from a table for which you do not have the `DELETE` privilege, the Oracle server does not permit the operation to take place.

If you receive the Oracle server error message "Table or view does not exist," you have done either of the following:

- Named a table or view that does not exist
- Attempted to perform an operation on a table or view for which you do not have the appropriate privilege

The data dictionary is organized in tables and views and contains information about the database. You can access the data dictionary to view the privileges that you have. The table in the slide describes various data dictionary views.

You learn about data dictionary views in the lesson titled "Introduction to Data Dictionary Views."

Note: The `ALL_TAB_PRIVS_MADE` dictionary view describes all the object grants made by the user or made on the objects owned by the user.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- System privileges
- Creating a role
- Object privileges
- Revoking object privileges

21

In this section, you will learn how to revoke object privileges.

Oracle Internal & Oracle Academy Use Only

Revoking Object Privileges

```
REVOKE {privilege [, privilege...]|ALL}
ON      object
FROM   {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

22

You can remove privileges granted to other users by using the `REVOKE` statement.

When you use the `REVOKE` statement, the privileges that you specify are revoked from the users you name and from any other users to whom those privileges were granted by the revoked user.

In the syntax:

`CASCADE CONSTRAINTS` Is required to remove any referential integrity constraints made to the object by means of the `REFERENCES` privilege

For more information, see the *Oracle Database SQL Language Reference* for Oracle Database19c.

Note: If a user leaves the company and you revoke his or her privileges, you must re-grant any privileges that this user granted to other users. If you drop the user account without revoking privileges from it, the system privileges granted by this user to other users are not affected by this action.

Oracle Internal & Oracle Academy Use Only

Revoking Object Privileges

Revoke the SELECT and INSERT privileges given to the demo user on the DEPARTMENTS table.

```
REVOKE  select, insert
ON      departments
FROM    demo;
```

```
Revoke succeeded.
```

23

The example in the slide revokes SELECT and INSERT privileges given to the demo user on the DEPARTMENTS table.

Note: If a user is granted a privilege with the WITH GRANT OPTION clause, that user can also grant the privilege with the WITH GRANT OPTION clause, so that a long chain of grantees is possible, but no circular grants (granting to a grant ancestor) are permitted. If the owner revokes a privilege from a user who granted the privilege to other users, the revoking cascades to all the privileges granted.

For example, if user A grants a SELECT privilege on a table to user B including the WITH GRANT OPTION clause, user B can grant to user C the SELECT privilege with the WITH GRANT OPTION clause as well, and user C can then grant to user D the SELECT privilege. If user A revokes privileges from user B, the privileges granted to users C and D are also revoked.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Differentiate system privileges from object privileges
- Grant privileges on tables
- Grant roles
- Distinguish between privileges and roles

24

DBAs establish initial database security for users by assigning privileges to the users.

- The DBA creates users who must have a password. The DBA is also responsible for establishing the initial system privileges for a user.
- After the user has created an object, the user can pass along any of the available object privileges to other users or to all users by using the `GRANT` statement.
- A DBA can create roles by using the `CREATE ROLE` statement to pass along a collection of system or object privileges to multiple users. Roles make granting and revoking privileges easier to maintain.
- Users can change their passwords by using the `ALTER USER` statement.
- You can remove privileges from users by using the `REVOKE` statement.
- With data dictionary views, users can view the privileges granted to them and those that are granted on their objects.

Oracle Internal & Oracle Academy Use Only

Practice 18: Overview

This practice covers the following topics:

- Granting privileges to other users on your table
- Modifying another user's table through the privileges granted to you

25

In this practice, you learn how to grant privileges to other users on your table and modifying another user's table through the privileges granted to you.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

19

Manipulating Data Using Advanced Queries

Oracle Internal & Oracle Academy Use Only

Course Roadmap

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

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to:

- Specify explicit default values in the `INSERT` and `UPDATE` statements
- Describe the features of multitable `INSERT`s
- Use the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
  - Pivoting `INSERT`
- Merge rows in a table
- Perform flashback operations
- Track the changes made to data over a period of time

3

In this lesson, you learn how to use the `DEFAULT` keyword in `INSERT` and `UPDATE` statements to identify a default column value. You also learn about multitable `INSERT` statements, the `MERGE` statement, performing flashback operations, and tracking changes in the database.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
  - Pivoting `INSERT`
- Merging rows in a table
- Performing flashback operations
- Tracking the changes in data over a period of time

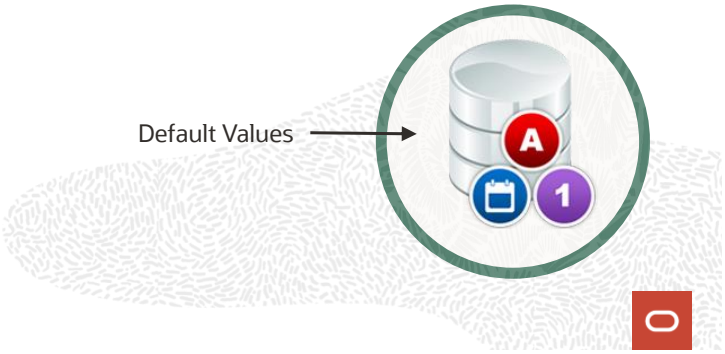
4

This section discusses explicit default values in `INSERT` and `UPDATE` statements.

Oracle Internal & Oracle Academy Use Only

Explicit Default Feature: Overview

- Use the `DEFAULT` keyword as a column value where the default column value is desired.
- This allows the user to control where and when the default value should be applied to data.
- Explicit defaults can be used in `INSERT` and `UPDATE` statements.



5

You can use the `DEFAULT` keyword in `INSERT` and `UPDATE` statements to identify a default column value. If no default value exists, a null value is used.

The `DEFAULT` option saves you from having to hard-code the default value in your programs or query the dictionary to find it, as was done before this feature was introduced. Hard-coding the default is a problem if the default changes, because the code consequently needs changing. Accessing the dictionary is not usually done in an application; therefore, this is a very important feature.

Oracle Internal & Oracle Academy Use Only

Using Explicit Default Values

- `DEFAULT` with `INSERT`:

```
INSERT INTO deptm3
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- `DEFAULT` with `UPDATE`:

```
UPDATE deptm3
SET manager_id = DEFAULT
WHERE department_id = 10;
```

6

Specify `DEFAULT` to set the column to the value previously specified as the default value for the column. If no default value for the corresponding column has been specified, the Oracle server sets the column to null.

In the first example in the slide, the `INSERT` statement uses a default value for the `MANAGER_ID` column. If there is no default value defined for the column, a null value is inserted.

The second example uses the `UPDATE` statement to set the `MANAGER_ID` column to a default value for department 10. If no default value is defined for the column, it changes the value to null.

Note: When creating a table, you can specify a default value for a column. This is discussed in *Lesson 11* of the course.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
  - Pivoting `INSERT`
- Merging rows in a table
- Performing flashback operations
- Tracking the changes in data over a period of time

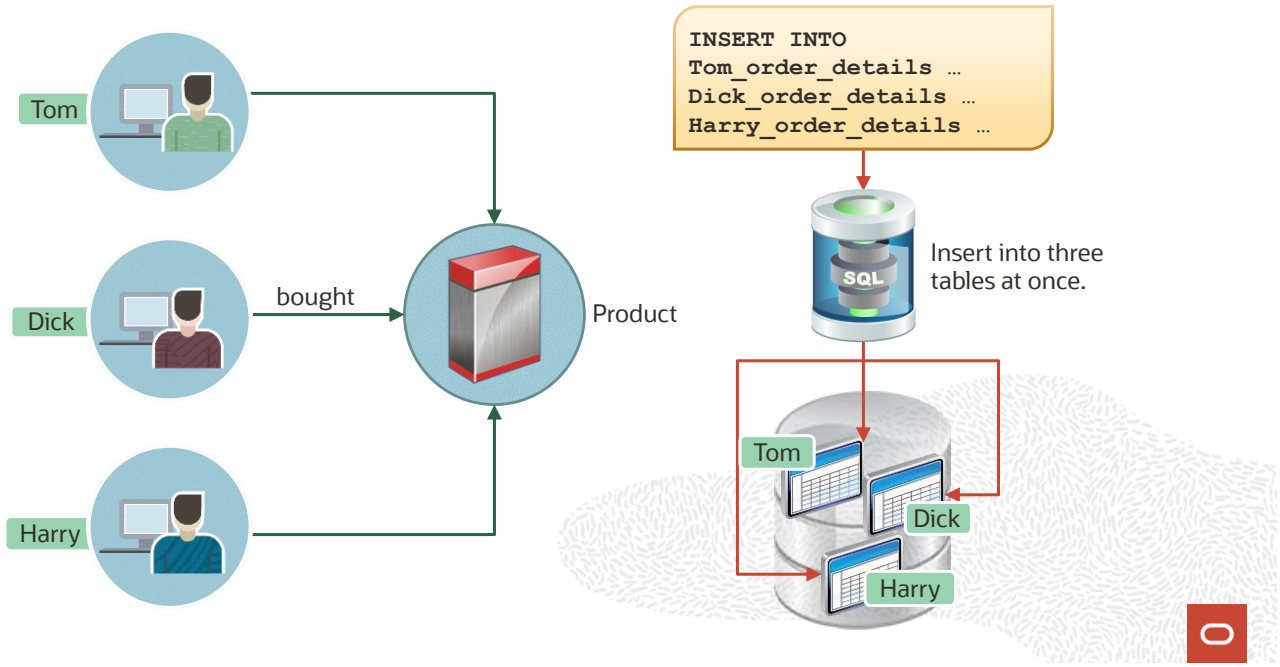
7

This section discusses the following types of multitable `INSERT`s:

- Unconditional `INSERT`
- Conditional `INSERT ALL`
- Conditional `INSERT FIRST`
- Pivoting `INSERT`

Oracle Internal & Oracle Academy Use Only

E-Commerce Scenario



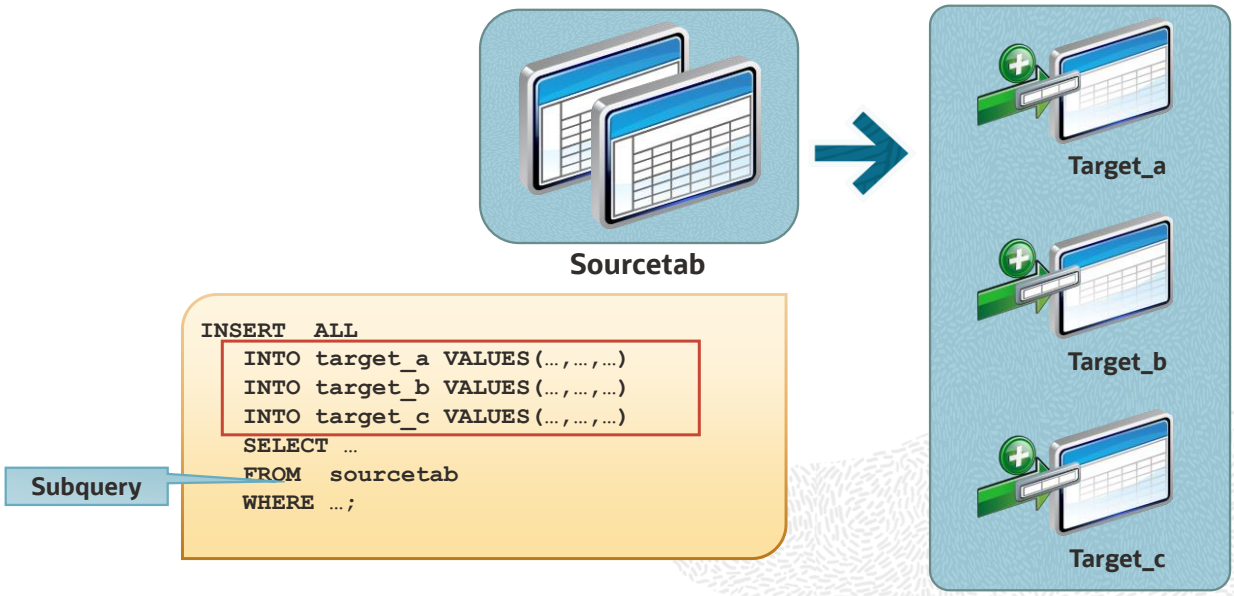
8

Recall the e-commerce company called OracleKart. Imagine Tom, Dick, and Harry are three customers who are shopping simultaneously from different parts of the globe. Coincidentally, the three of them add the same item into their shopping carts. They check out and make the final payment for the product. At this point, an entry has to be made for the order details in each of their customer accounts.

Executing three different `INSERT` statements will hinder the performance of the database. In such a scenario, we can use a multitable `INSERT` statement to simultaneously make an entry in three of the `CUSTOMER` tables.

Oracle Internal & Oracle Academy Use Only

Multitable INSERT Statements: Overview



9

In a multitable INSERT statement, you insert computed rows into one or more tables. These computed rows are derived from the rows returned from the evaluation of a subquery.

Multitable INSERT statements are useful in a data warehouse scenario. You need to load your data warehouse regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational systems must be extracted and copied into the warehouse. The process of extracting data from the source system and bringing it into the data warehouse is commonly called ETL, which stands for extraction, transformation, and loading.

During extraction, the desired data must be identified and extracted from many different sources, such as database systems and applications. After extraction, the data must be physically transported to the target system or an intermediate system for further processing. Depending on the chosen means of transportation, some transformations can be done during this process. For example, a SQL statement that directly accesses a remote target through a gateway can concatenate two columns as part of the SELECT statement.

After data is loaded into the Oracle database, data transformations can be executed using SQL operations. A multitable INSERT statement is one of the techniques for implementing SQL data transformations.

Oracle Internal & Oracle Academy Use Only

Multitable INSERT Statements: Overview

- Use the INSERT…SELECT statement to insert rows into multiple tables as a part of a single DML statement.
- Multitable INSERT statements are used in data warehousing systems to transfer data from one or more operational sources to a set of target tables.
- They provide significant performance improvement over:
  - Single DML versus multiple INSERT…SELECT statements
  - Single DML versus a procedure to perform multiple inserts by using the IF…THEN syntax

10

Multitable INSERT statements offer the benefits of the INSERT...SELECT statement when multiple tables are involved as targets.

Without multitable INSERT, you had to deal with *n* independent INSERT...SELECT statements, thus processing the same source data *n* times and increasing the transformation workload *n* times.

As with the existing INSERT...SELECT statement, you can use the new statement in parallel with direct-load mechanism for faster performance.

Each record from any input stream, such as a nonrelational database table, can now be converted into multiple records for a more relational database table environment. To alternatively implement this functionality, you were required to write multiple INSERT statements.

Oracle Internal & Oracle Academy Use Only

Types of Multitable INSERT Statements

Different types of multitable `INSERT` statements:

- Unconditional `INSERT`
- Conditional `INSERT ALL`
- Conditional `INSERT FIRST`
- Pivoting `INSERT`

11

You use different clauses to indicate the type of `INSERT` to be executed. The types of multitable `INSERT` statements are:

- **Unconditional `INSERT`:** For each row returned by the subquery, a row is inserted into each of the target tables.
- **Conditional `INSERT ALL`:** For each row returned by the subquery, a row is inserted into each target table if the specified condition is met.
- **Conditional `INSERT FIRST`:** For each row returned by the subquery, a row is inserted into the very first target table in which the condition is met.
- **Pivoting `INSERT`:** This is a special case of the unconditional `INSERT ALL`.

Oracle Internal & Oracle Academy Use Only

Multitable INSERT Statements

- Syntax for multitable INSERT:

```
INSERT [ ALL
{ insert_into_clause [ values_clause ]}...
| conditional_insert_clause
 ] subquery
```

- Conditional_insert_clause:

```
INSERT [ ALL | FIRST ]
 WHEN condition THEN insert_into_clause
                       [ values_clause ]
                       [ insert_into_clause [ values_clause ]]…
  [ ELSE insert_into_clause
                       [ values_clause ]
                       [ insert_into_clause [ values_clause ]]…
 ]
```

12

The slide displays the generic format for multitable INSERT statements.

Unconditional INSERT: ALL into_clause

Specify ALL followed by multiple insert_into_clauses to perform an unconditional multitable INSERT. The Oracle Server executes each insert_into_clause once for each row returned by the subquery.

Conditional INSERT: conditional_insert_clause

Specify the conditional_insert_clause to perform a conditional multitable INSERT. The Oracle Server filters each insert_into_clause through the corresponding WHEN condition, which determines whether that insert_into_clause is executed. A single multitable INSERT statement can contain up to 127 WHEN clauses.

Conditional INSERT: ALL

If you specify ALL, the Oracle Server evaluates each WHEN clause regardless of the results of the evaluation of any other WHEN clause. For each WHEN clause whose condition evaluates to true, the Oracle Server executes the corresponding INTO clause list.

Conditional INSERT: FIRST

If you specify FIRST, the Oracle Server evaluates each WHEN clause in the order in which it appears in the statement. If the first WHEN clause evaluates to true, the Oracle Server executes the corresponding INTO clause and skips subsequent WHEN clauses for the given row.

Oracle Internal & Oracle Academy Use Only

Conditional `INSERT: ELSE` Clause

For a given row, if no `WHEN` clause evaluates to true:

- If you have specified an `ELSE` clause, the Oracle Server executes the `INTO` clause list associated with the `ELSE` clause
- If you did not specify an `ELSE` clause, the Oracle Server takes no action for that row

Restrictions on Multitable `INSERT` Statements

- You can perform multitable `INSERT` statements only on tables, and not on views or materialized views.
- You cannot perform a multitable `INSERT` on a remote table.
- You cannot specify a table collection expression when performing a multitable `INSERT`.
- In a multitable `INSERT`, all `insert_into_clauses` cannot combine to specify more than 999 target columns.

Oracle Internal & Oracle Academy Use Only

Unconditional INSERT ALL

- Select the EMPLOYEE_ID, HIRE_DATE, SALARY, and MANAGER_ID values from the EMPLOYEES table for those employees whose EMPLOYEE_ID is greater than 200.
- Insert these values into the SAL_HISTORY and MGR_HISTORY tables by using a multitable INSERT.

```
INSERT  ALL
   INTO sal_history VALUES(EMPID,HIREDATE,SAL)
   INTO mgr_history VALUES(EMPID,MGR,SAL)
   SELECT employee_id EMPID, hire_date HIREDATE,
          salary SAL, manager_id MGR
   FROM  employees
   WHERE employee_id > 200;
```

```
12 rows inserted
```

14

The example in the slide inserts rows into both the SAL_HISTORY and the MGR_HISTORY tables.

- The SELECT statement retrieves the details of employee ID, hire date, salary, and manager ID of those employees whose employee ID is greater than 200 from the EMPLOYEES table.
- The details of the employee ID, hire date and salary are inserted into the SAL_HISTORY table.
- The details of employee ID, manager ID, and salary are inserted into the MGR_HISTORY table.

This INSERT statement is referred to as an unconditional INSERT because no further restriction is applied to the rows that are retrieved by the SELECT statement. All the rows retrieved by the SELECT statement are inserted into the two tables: SAL_HISTORY and MGR_HISTORY. The VALUES clause in the INSERT statements specifies the columns from the SELECT statement that must be inserted into each of the tables. Each row returned by the SELECT statement results in two insertions: one for the SAL_HISTORY table and one for the MGR_HISTORY table.

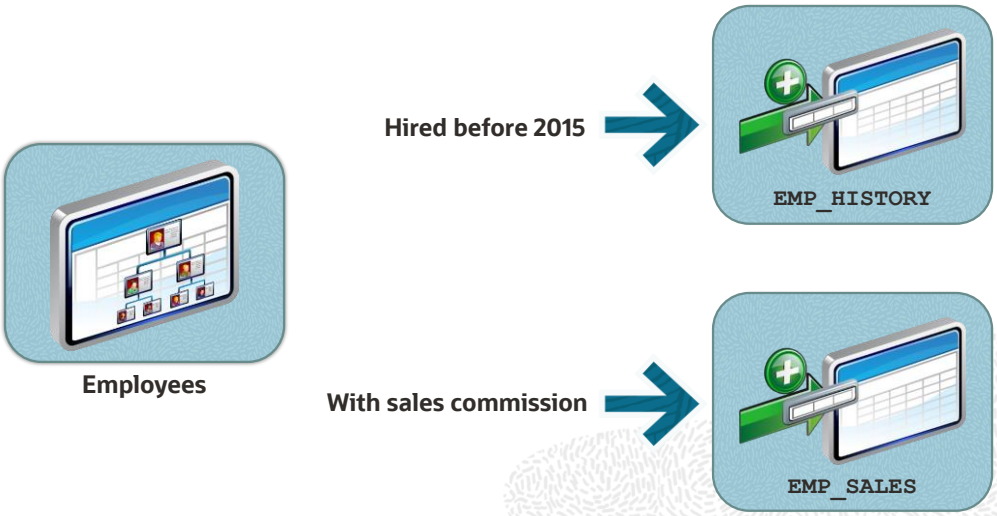
A total of 12 rows were inserted:

```
SELECT COUNT(*) total_in_sal FROM sal_history;

SELECT COUNT(*) total_in_mgr FROM mgr_history;
```

Oracle Internal & Oracle Academy Use Only

Conditional INSERT ALL: Example



15

For all employees in the employees tables, if the employee was hired before 2015, insert that employee record into employee history. If the employee earns a sales commission, insert the record information into the EMP_SALES table. The SQL statement is shown in the next slide.

Oracle Internal & Oracle Academy Use Only

Conditional INSERT ALL

```
INSERT  ALL
 WHEN HIREDATE < '01-JAN-15' THEN
   INTO emp_history VALUES(EMPID,HIREDATE,SAL)
 WHEN COMM IS NOT NULL THEN
   INTO emp_sales VALUES(EMPID,COMM,SAL)
   SELECT employee_id EMPID, hire_date HIREDATE,
          salary SAL, commission_pct COMM
   FROM  employees;
```

```
112 rows inserted.
```

16

The example in the slide is similar to the example in the previous slide because it inserts rows into both EMP_HISTORY and EMP_SALES tables.

The SELECT statement retrieves details such as employee ID, hire date, salary, and commission percentage for all employees from the EMPLOYEES table. Details such as employee ID, hire date, and salary are inserted into the EMP_HISTORY table. Details such as employee ID, commission percentage, and salary are inserted into the EMP_SALES table.

This INSERT statement is referred to as a conditional INSERT ALL because a further restriction is applied to the rows that are retrieved by the SELECT statement. From the rows that are retrieved by the SELECT statement, only those rows in which the hire date was prior to 2015 are inserted in the EMP_HISTORY table. Similarly, only those rows where the value of commission percentage is not null are inserted in the EMP_SALES table.

```
SELECT count(*) FROM emp_history;
```

Result: 77 rows fetched.

```
SELECT count(*) FROM emp_sales;
```

Result: 35 rows fetched.

Oracle Internal & Oracle Academy Use Only

You can also optionally use the `ELSE` clause with the `INSERT ALL` statement.

Example:

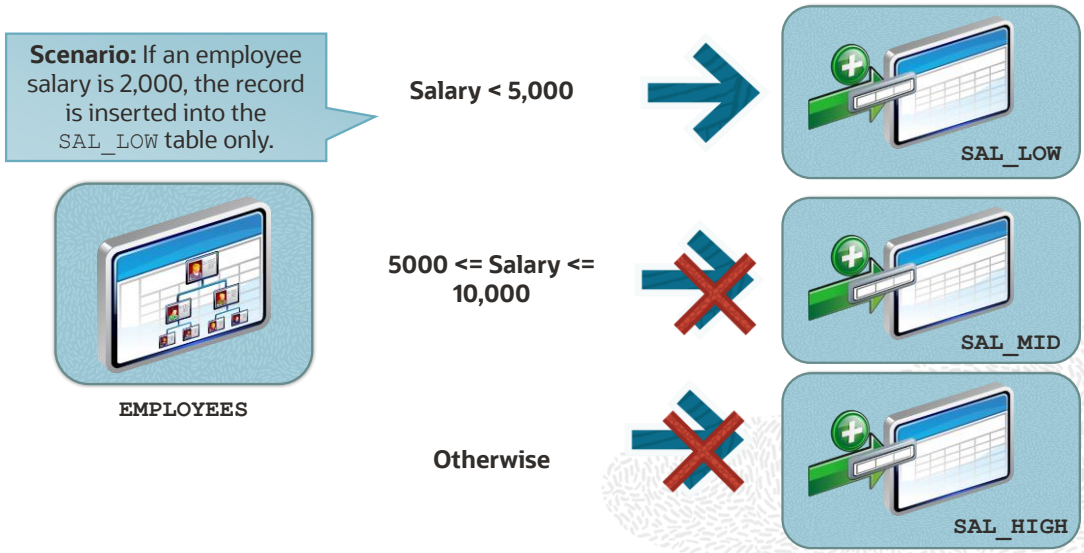
```
INSERT ALL
WHEN job_id IN
(select job_id FROM jobs WHERE job_title LIKE '%Manager%') THEN
INTO managers2(last_name,job_id,SALARY)
VALUES (last_name,job_id,SALARY)
WHEN SALARY>10000 THEN
INTO richpeople(last_name,job_id,SALARY)
VALUES (last_name,job_id,SALARY)
ELSE
INTO poorpeople VALUES (last_name,job_id,SALARY)
SELECT * FROM employees;
```

Result:

```
116 rows inserted
```

Oracle Internal & Oracle Academy Use Only

Conditional INSERT FIRST: Example



18

For all employees in the EMPLOYEES table, insert the employee information into the first target table that meets the condition. In the example, if an employee has a salary of 2,000, the record is inserted into the SAL_LOW table only. The SQL statement is shown in the next slide.

Oracle Internal & Oracle Academy Use Only

Conditional INSERT FIRST

```
INSERT FIRST
WHEN salary < 5000 THEN
  INTO sal_low VALUES (employee_id, last_name, salary)
WHEN salary between 5000 and 10000 THEN
  INTO sal_mid VALUES (employee_id, last_name, salary)
ELSE
  INTO sal_high VALUES (employee_id, last_name, salary)
SELECT employee_id, last_name, salary
FROM employees;
```

```
107 rows inserted
```

19

The SELECT statement retrieves details such as employee ID, last name, and salary for every employee in the EMPLOYEES table. For each employee record, it is inserted into the very first target table that meets the condition.

This INSERT statement is referred to as a conditional INSERT FIRST. The WHEN salary < 5000 condition is evaluated first. If this first WHEN clause evaluates to true, the Oracle Server executes the corresponding INTO clause and inserts the record into the SAL_LOW table. It skips subsequent WHEN clauses for this row.

If the row does not satisfy the first WHEN condition (WHEN salary < 5000), the next condition (WHEN salary between 5000 and 10000) is evaluated. If this condition evaluates to true, the record is inserted into the SAL_MID table, and the last condition is skipped.

If neither the first condition (WHEN salary < 5000) nor the second condition (WHEN salary between 5000 and 10000) is evaluated to true, the Oracle Server executes the corresponding INTO clause for the ELSE clause.

A total of 107 rows were inserted:

```
SELECT count(*) low FROM sal_low;
49 rows fetched.
SELECT count(*) mid FROM sal_mid;
43 rows fetched.
SELECT count(*) high FROM sal_high;
15 rows fetched.
```

Oracle Internal & Oracle Academy Use Only

Pivoting INSERT

Convert the set of sales records from the nonrelational database table to the relational format.

Emp_ID	Week_ID	MON	TUES	WED	THUR	FRI
176	6	2000	3000	4000	5000	6000

Employee_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000

20

Pivoting is an operation in which you must build a transformation such that each record from any input stream, such as a nonrelational database table, must be converted into multiple records for a more relational database table environment.

Suppose you receive a set of sales records from a nonrelational database table: `SALES_SOURCE_DATA`, in the following format:

```
EMPLOYEE_ID, WEEK_ID, SALES_MON, SALES_TUE, SALES_WED, SALES_THUR,
SALES_FRI
```

You want to store these records in the `SALES_INFO` table in a more typical relational format:

```
EMPLOYEE_ID, WEEK, SALES
```

To solve this problem, you must build a transformation such that each record from the original nonrelational database table, `SALES_SOURCE_DATA`, is converted into five records for the data warehouse's `SALES_INFO` table. This operation is commonly referred to as *pivoting*.

The solution to this problem is shown in the next slide.

Oracle Internal & Oracle Academy Use Only

Pivoting INSERT

```
INSERT ALL
  INTO sales_info VALUES (employee_id,week_id,sales_MON)
  INTO sales_info VALUES (employee_id,week_id,sales_TUE)
  INTO sales_info VALUES (employee_id,week_id,sales_WED)
  INTO sales_info VALUES (employee_id,week_id,sales_THUR)
  INTO sales_info VALUES (employee_id,week_id, sales_FRI)
  SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
         sales_WED, sales_THUR,sales_FRI
  FROM sales_source_data;
```

```
5 rows inserted
```

21

In the example, the sales data is received from the nonrelational database table SALES_SOURCE_DATA, which has the details of the sales performed by a sales representative on each day of a week, for a week with a particular week ID.

```
DESC SALES_SOURCE_DATA
```

```
DESC SALES_SOURCE_DATA
Name        Null Type
----------- ---- -----------
EMPLOYEE_ID      NUMBER(6)
WEEK_ID          NUMBER(2)
SALES_MON        NUMBER(8,2)
SALES_TUE        NUMBER(8,2)
SALES_WED        NUMBER(8,2)
SALES_THUR       NUMBER(8,2)
SALES_FRI        NUMBER(8,2)
```

```
SELECT * FROM SALES_SOURCE_DATA;
```

	EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
1	178	6	1750	2200	1500	1500	3000

Oracle Internal & Oracle Academy Use Only

```
DESC SALES_INFO
```

```
desc sales_info
Name        Null Type
----------- ---- -----------
EMPLOYEE_ID      NUMBER(6)
WEEK             NUMBER(2)
SALES            NUMBER(8,2)
```

```
SELECT * FROM sales_info;
```

	EMPLOYEE_ID	WEEK	SALES
1	178	6	1750
2	178	6	2200
3	178	6	1500
4	178	6	1500
5	178	6	3000

Observe in the preceding example that by using a pivoting `INSERT`, one row from the `SALES_SOURCE_DATA` table is converted into five records for the relational table, `SALES_INFO`.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
  - Pivoting `INSERT`
- Merging rows in a table
- Performing flashback operations
- Tracking the changes in data over a period of time

23

This section discusses merging rows in a table.

Oracle Internal & Oracle Academy Use Only

MERGE Statement

- Provides the ability to conditionally update, insert, or delete data into a database table
- Performs an `UPDATE` if the row exists and an `INSERT` if it is a new row:
  - Avoids separate updates
  - Increases performance and ease of use
  - Is useful in data warehousing applications

24

The `MERGE` statement is suitable in a number of data warehousing applications. For example, in a data warehousing application, you may need to work with data coming from multiple sources, some of which may be duplicate. With the `MERGE` statement, you can conditionally add or modify rows.

The Oracle Server supports the `MERGE` statement for `INSERT`, `UPDATE`, and `DELETE` operations. By using this statement, you can update, insert, or delete a row conditionally into a table, thus avoiding multiple DML statements. The decision whether to update, insert, or delete into the target table is based on a condition in the `ON` clause.

You must have the `INSERT` and `UPDATE` object privileges on the target table and the `SELECT` object privilege on the source table. To specify the `DELETE` clause of `merge_update_clause`, you must also have the `DELETE` object privilege on the target table.

The `MERGE` statement is deterministic. You cannot update the same row of the target table multiple times in the same `MERGE` statement.

An alternative approach is to use PL/SQL loops and multiple DML statements. The `MERGE` statement, however, is easy to use and more simply expressed as a single SQL statement.

Oracle Internal & Oracle Academy Use Only

MERGE Statement Syntax

You can conditionally insert, update, or delete rows in a table by using the MERGE statement.

```
MERGE INTO table_name table_alias
  USING (table|view|sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
    col1 = col1_val,
    col2 = col2_val
 WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

25

Merging Rows

You can update existing rows and insert new rows conditionally by using the MERGE statement. Using the MERGE statement, you can delete obsolete rows at the same time as you update rows in a table. To do this, you include a DELETE clause with its own WHERE clause in the syntax of the MERGE statement.

In the syntax:

INTO clause	Specifies the target table you are updating or inserting into
USING clause or subquery	Identifies the source of the data to be updated or inserted; can be a table, view,
ON clause	The condition on which the MERGE operation either updates or inserts
WHEN MATCHED | WHEN NOT MATCHED	Instructs the server how to respond to the results of the join condition

Note: For more information, see *Oracle Database SQL Language Reference* for Oracle Database 19c.

Oracle Internal & Oracle Academy Use Only

Merging Rows: Example

Insert or update rows in the COPY_EMP3 table to match the EMPLOYEES table.

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
...

DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

```
107 rows merged.
```

26

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
c.email = e.email,
c.phone_number = e.phone_number,
c.hire_date = e.hire_date,
c.job_id = e.job_id,
c.salary = e.salary*2,
c.commission_pct = e.commission_pct,
c.manager_id = e.manager_id,
c.department_id = e.department_id
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
```

Oracle Internal & Oracle Academy Use Only

```
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

The COPY_EMP3 table is created by using the following code:

```
CREATE TABLE COPY_EMP3 AS SELECT * FROM EMPLOYEES
WHERE SALARY<10000;
```

Then query the COPY_EMP3 table.

```
SELECT employee_id, salary, commission_pct FROM COPY_EMP3;
```

Observe that in the output, there are some employees with SALARY < 10000 and there are two employees with COMMISSION_PCT.

The example in the slide matches the EMPLOYEE_ID in the COPY_EMP3 table to the EMPLOYEE_ID in the EMPLOYEES table. If a match is found, the row in the COPY_EMP3 table is updated to match the row in the EMPLOYEES table and the salary of the employee is doubled. The records of the two employees with values in the COMMISSION_PCT column are deleted. If the match is not found, rows are inserted into the COPY_EMP3 table.

Oracle Internal & Oracle Academy Use Only

Merging Rows: Example

```
TRUNCATE TABLE copy_emp3;
SELECT * FROM copy_emp3;
no rows selected
```

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
...
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, ...
```

```
SELECT * FROM copy_emp3;
107 rows selected.
```

28

The examples in the slide show that the `COPY_EMP3` table is empty. The `c.employee_id = e.employee_id` condition is evaluated. The condition returns false—there are no matches. The logic falls into the `WHEN NOT MATCHED` clause, and the `MERGE` command inserts the rows of the `EMPLOYEES` table into the `COPY_EMP3` table. This means that the `COPY_EMP3` table now has exactly the same data as in the `EMPLOYEES` table.

```
SELECT employee_id, salary, commission_pct from copy_emp3;
```

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
  - Pivoting `INSERT`
- Merging rows in a table
- Performing flashback operations
- Tracking the changes in data over a period of time

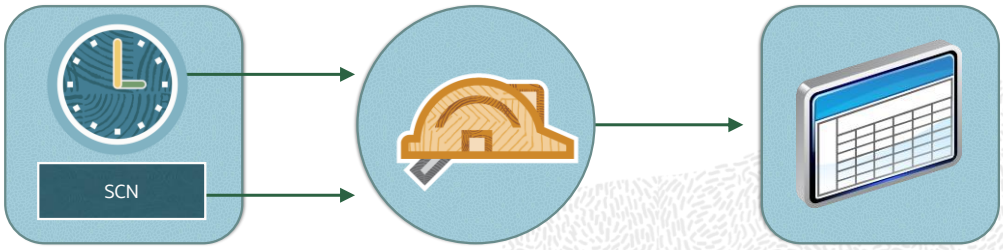
29

This section discusses performing flashback operations.

Oracle Internal & Oracle Academy Use Only

FLASHBACK TABLE Statement

- Enables you to recover tables to a specified point in time with a single statement
- Restores table data along with associated indexes and constraints
- Enables you to revert the table and its contents to a certain point in time or system change number (SCN)



30

Oracle Flashback Table enables you to recover tables to a specified point in time with a single statement. You can restore table data along with associated indexes and constraints while the database is online, undoing changes to only the specified tables.

The Flashback Table feature is similar to a self-service repair tool. For example, if a user accidentally deletes important rows from a table and then wants to recover the deleted rows, you can use the `FLASHBACK TABLE` statement to restore the table to the time before the deletion and see the missing rows in the table.

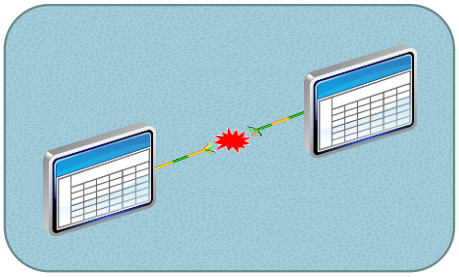
When using the `FLASHBACK TABLE` statement, you can revert the table and its contents to a certain time or to an SCN.

Note: The SCN is an integer value associated with each change to the database. It is a unique incremental number in the database. Every time you commit a transaction, a new SCN is recorded.

Oracle Internal & Oracle Academy Use Only

FLASHBACK TABLE Statement

- Is a repair tool for accidental table modifications
- Restores a table to an earlier point in time
- Offers ease of use, availability, and fast execution
- Is performed in place
- Syntax:



```
FLASHBACK TABLE [ schema. ] table [, [ schema. ] table ]... TO { { {
SCN | TIMESTAMP } expr | RESTORE POINT restore_point } [ { ENABLE |
DISABLE } TRIGGERS ] | BEFORE DROP [ RENAME TO table ] } ;
```

31

Self-Service Repair Facility

You can use the SQL data definition language (DDL) command, `FLASHBACK TABLE`, provided by Oracle Database to restore the state of a table to an earlier point in time, in case it is inadvertently deleted or modified.

The `FLASHBACK TABLE` command is a self-service repair tool to restore data in a table along with associated attributes such as indexes or views. This is done, while the database is online, by rolling back only the subsequent changes to the given table. Compared to traditional recovery mechanisms, this feature offers significant benefits such as ease of use, availability, and faster restoration. It also takes the burden off the DBA to find and restore application-specific properties. The flashback table feature does not address physical corruption caused because of a bad disk.

Syntax

You can invoke a `FLASHBACK TABLE` operation on one or more tables, even on tables in different schemas. You specify the point in time to which you want to revert by providing a valid time stamp. By default, database triggers are disabled during the flashback operation for all tables involved. You can override this default behavior by specifying the `ENABLE TRIGGERS` clause.

Note: For more information about recycle bin and flashback semantics, refer to *Oracle Database Administrator's Guide* for Oracle Database 19c.

Oracle Internal & Oracle Academy Use Only

Using the FLASHBACK TABLE Statement

```
DROP TABLE emp3;
```

```
Table EMP3 dropped.
```

```
SELECT original_name, operation, droptime FROM recyclebin;
```

ORIGINAL_NAME	OPERATION	DROPTIME
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
7 EMP3	DROP	2016-08-26:01:53:10

```
FLASHBACK TABLE emp3 TO BEFORE DROP;
```

```
Flashback succeeded.
```

32

Syntax and Examples

The example restores the `EMP3` table to a state before a `DROP` statement.

You can query the recycle bin data dictionary table to fetch information about dropped objects. Dropped tables and any associated objects—such as, indexes, constraints, nested tables, and so on—are not removed and still occupy space. They continue to count against user space quotas until you specifically purge from the recycle bin, or until they must be purged by the database because of tablespace space constraints.

Each user can be thought of as an owner of a recycle bin because, unless a user has the `SYSDBA` privilege, the only objects that the user has access to in the recycle bin are those that the user owns. A user can view his or her objects in the recycle bin by using the following statement:

```
SELECT * FROM RECYCLEBIN;
```

When you drop a user, objects belonging to that user are not placed in the recycle bin and those in the recycle bin are purged.

You can purge the recycle bin with the following statement:

```
PURGE RECYCLEBIN;
```

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

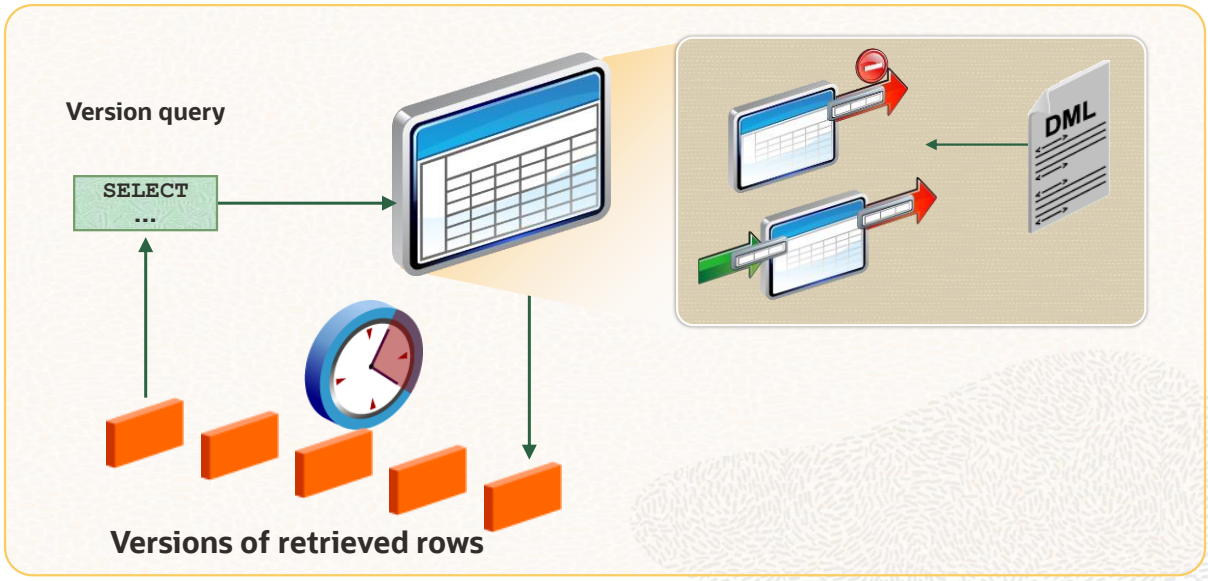
- Specifying explicit default values in the `INSERT` and `UPDATE` statements
- Using the following types of multitable `INSERT`s:
  - Unconditional `INSERT`
  - Conditional `INSERT ALL`
  - Conditional `INSERT FIRST`
  - Pivoting `INSERT`
- Merging rows in a table
- Performing flashback operations
- Tracking the changes in data over a period of time

33

In this section, you will learn how to track changes in data over a period of time.

Oracle Internal & Oracle Academy Use Only

Tracking Changes in Data



34

You may discover that, somehow, data in a table has been inappropriately changed. To research this, you can use multiple flashback queries to view row data at specific points in time. You can use Oracle Flashback Query to retrieve data as it existed at an earlier time.

More efficiently, you can use the Flashback Version Query feature to view all the changes to a row over a period of time. This feature enables you to append a `VERSIONS` clause to a `SELECT` statement that specifies an SCN or the time-stamp range within which you want to view changes to row values. The query also can return associated metadata, such as the transaction responsible for the change.

Further, after you identify an erroneous transaction, you can use the Flashback Transaction Query feature to identify other changes that were done by the transaction. You then have the option of using the Flashback Table feature to restore the table to a state before the changes were made.

You can use a query on a table with a `VERSIONS` clause to produce all the versions of all the rows that exist, or ever existed, between the time the query was issued and the `undo_retention` seconds before the current time. `undo_retention` is an initialization parameter, which is an auto-tuned parameter.

A query that includes a `VERSIONS` clause is referred to as a version query. The results of a version query behave as though the `WHERE` clause were applied to the versions of the rows. The version query returns versions of the rows only across transactions.

SCN: The Oracle server assigns an SCN to identify the redo records for each committed transaction.

Oracle Internal & Oracle Academy Use Only

Flashback Query: Example

```
SELECT salary FROM employees3
WHERE last_name = 'Chung';
```

```
UPDATE employees3 SET salary = 4000
WHERE last_name = 'Chung';

SELECT salary FROM employees3
WHERE last_name = 'Chung';
```

```
SELECT salary FROM employees3
AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' MINUTE)
WHERE last_name = 'Chung';
```

1

	SALARY
1	3800

2

	SALARY
1	4000

3

	SALARY
1	3800

35

To use Oracle Flashback Query, use a `SELECT` statement with an `AS OF` clause. Oracle Flashback Query retrieves data as it existed at a particular time in the past. The query explicitly references a past time through a timestamp or SCN. It returns committed data that was current at that point in time.

In the example in the slide, the salary for employee `Chung` is retrieved (1). The salary for employee `Chung` is increased to 4000 (2). To learn what the value was before the update, you can use the Flashback Query(3).

Oracle Flashback Query can be used in the following scenarios:

- To recover lost data or to undo incorrect, committed changes. For example, if you mistakenly delete or update rows, and then commit them, you can immediately undo the mistake.
- To compare current data with the corresponding data at some time in the past. For example, you can run a daily report that shows the change in data from yesterday. You can compare individual rows of table data or find intersections or unions of sets of rows.
- To check the state of transactional data at a particular time

Oracle Internal & Oracle Academy Use Only

Flashback Version Query: Example

```
SELECT salary FROM employees3
WHERE  employee_id = 107;
```

1

```
UPDATE employees3 SET salary = salary * 1.30
WHERE  employee_id = 107;

COMMIT;
```

2

```
SELECT salary FROM employees3
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 107;
```

3

1

	SALARY
1	4200

3

	SALARY
1	5460
2	4200

36

In the example in the slide, the salary for employee 107 is retrieved (1). The salary for employee 107 is increased by 30 percent and this change is committed (2). The different versions of salary are displayed (3).

The `VERSIONS` clause does not change the plan of the query. For example, if you run a query on a table that uses the index access method, the same query on the same table with a `VERSIONS` clause continues to use the index access method.

The versions of the rows returned by the version query are versions of the rows across transactions. The `VERSIONS` clause has no effect on the transactional behavior of a query. This means that a query on a table with a `VERSIONS` clause still inherits the query environment of the ongoing transaction.

The default `VERSIONS` clause can be specified as `VERSIONS BETWEEN {SCN|TIMESTAMP} MINVALUE AND MAXVALUE`. The `VERSIONS` clause is a SQL extension only for queries. You can have DML and DDL operations that use a `VERSIONS` clause within subqueries. The row version query retrieves all the committed versions of the selected rows. Changes made by the current active transaction are not returned. The version query retrieves all incarnations of the rows. This essentially means that versions returned include deleted and subsequent reinserted versions of the rows. The row access for a version query can be defined in one of the following two categories:

- **ROWID-based row access:** In case of `ROWID`-based access, all versions of the specified `ROWID` are returned irrespective of the row content. This essentially means that all versions of the slot in the block indicated by the `ROWID` are returned.
- **All other row access:** For all other row access, all versions of the rows are returned.

Oracle Internal & Oracle Academy Use Only

VERSIONS BETWEEN Clause

```
SELECT versions_starttime "START_DATE",
       versions_endtime   "END_DATE",
       salary
FROM   employees
    VERSIONS BETWEEN SCN MINVALUE
    AND MAXVALUE
WHERE  last_name = 'Lorentz';
```

	START_DATE	END_DATE	SALARY
1	26-AUG-16 03.00.07.000000000 AM	(null)	5460
2	(null)	26-AUG-16 03.00.07.000000000 AM	4200

```
SELECT salary FROM employees3
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE  employee_id = 107;
```

37

You can use the VERSIONS BETWEEN clause to retrieve all the versions of the rows that exist or have ever existed between the time the query was issued and a point back in time.

If the undo retention time is less than the lower bound time or the SCN of the BETWEEN clause, the query retrieves versions up to the undo retention time only. The time interval of the BETWEEN clause can be specified as an SCN interval or a wall-clock interval. This time interval is closed at both the lower and the upper bounds.

In the example, Lorentz's salary changes are retrieved. The NULL value for END_DATE for the first version indicates that this was the existing version at the time of the query. The NULL value for START_DATE for the last version indicates that this version was created at a time before the undo retention time.

Note: The START_DATE and END_DATE values in the screenshot vary according to when the query was executed.

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to:

- Specify explicit default values in the `INSERT` and `UPDATE` statements
- Describe the features of multitable `INSERT`s
- Use the following types of multitable `INSERT`s:
  - `Unconditional INSERT`
  - `Conditional INSERT ALL`
  - `Conditional INSERT FIRST`
  - `Pivoting INSERT`
- Merge rows in a table
- Perform flashback operations
- Track the changes in data over a period of time

38

In this lesson, you also should have learned about multitable `INSERT` statements, the `MERGE` statement, and tracking changes in the database.

Oracle Internal & Oracle Academy Use Only

Practice 19: Overview

This practice covers the following topics:

- Performing multitable `INSERT`s
- Performing `MERGE` operations
- Performing flashback operations
- Tracking row versions

39

In this practice, you learn how to perform multitable `INSERTS`, `MERGE` operations, flashback operations, and tracking row versions.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

20

Managing 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

You are here!

2

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

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this lesson, you should be able to use:

- Data types similar to DATE that store fractional seconds and track time zones
- Data types that store the difference between two datetime values
- The following datetime functions:
  - CURRENT_DATE
  - CURRENT_TIMESTAMP
  - LOCALTIMESTAMP
  - DBTIMEZONE
  - SESSIONTIMEZONE
  - EXTRACT
  - TZ_OFFSET
  - FROM_TZ
  - TO_TIMESTAMP
  - TO_YMINTERVAL
  - TO_DSINTERVAL

3

In this lesson, you learn how to use data types similar to DATE that store fractional seconds and track time zones. This lesson addresses some of the datetime functions available in the Oracle database.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

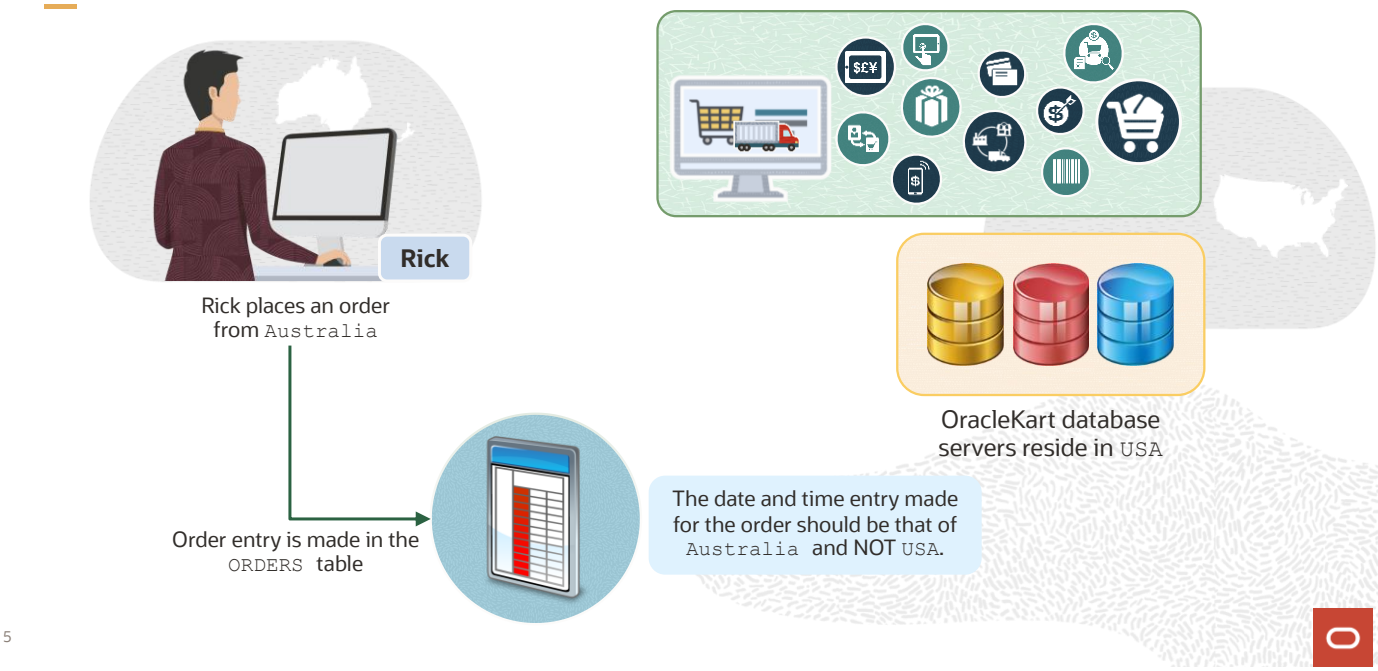
- CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP
- INTERVAL data types
- Using the following functions:
  - EXTRACT
  - TZ_OFFSET
  - FROM_TZ
  - TO_TIMESTAMP
  - TO_YMINTERVAL
  - TO_DSINTERVAL

4

This section discusses CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP.

Oracle Internal & Oracle Academy Use Only

E-Commerce Scenario

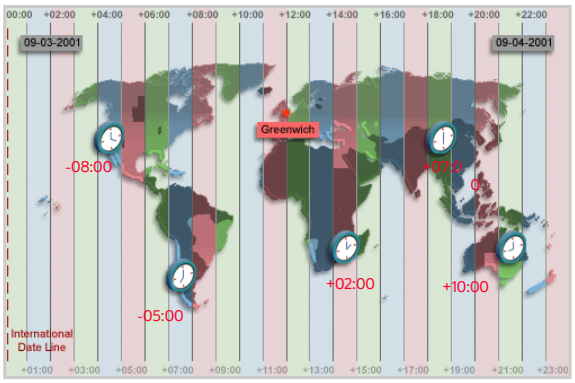


Recall the OracleKart e-commerce company that is a global online shopping website. Their data center is located in the US. The customers of this site are spread out all around the globe. When a customer, Rick, who lives in Australia, places an order, the date and time of the order is recorded as per the US time zone. Due to this, the order delivery date is miscalculated, and the order is delayed. How can this be avoided?

Ideally, the order date and time should be the local time of the place from which the customer placed the order (in this case, Australia). Nonetheless, if the time zone is not set, the date and time will be set to that of the server where the database resides. For example, if the database is in the US, then the date and time entry will be in the US time zone. This will cause problems while calculating the delivery time and returns for the customer. Hence, time zones are very useful for the e-commerce industry. Let us learn more about the different time zones and how to use them.

Oracle Internal & Oracle Academy Use Only

Time Zones



The image represents the time for each time zone when Greenwich time is 12:00.

6

Let us look into some of the basics of time zones:

The hours of the day are measured by the turning of the earth. The time of day at any particular moment depends on where you are. When it is noon in Greenwich, England, it is midnight along the International Date Line.

The earth is divided into 24 time zones, one for each hour of the day. The time along the prime meridian in Greenwich, England, is known as Greenwich Mean Time (GMT). GMT is now known as Coordinated Universal Time (UTC).

UTC is the time standard against which all other time zones in the world are referenced. It is the same all year round and is not affected by summer time or daylight saving time.

The meridian line is an imaginary line that runs from the North Pole to the South Pole. It is known as zero longitude, and it is the line from which all other lines of longitude are measured. All time is measured relative to UTC and all places have a latitude (their distance north or south of the equator) and a longitude (their distance east or west of the Greenwich meridian).

Oracle Internal & Oracle Academy Use Only

TIME_ZONE Session Parameter

`TIME_ZONE` may be set to:

- An absolute offset
- Database time zone
- OS local time zone
- A named region

```
ALTER SESSION SET TIME_ZONE = '-05:00';
ALTER SESSION SET TIME_ZONE = dbtimezone;
ALTER SESSION SET TIME_ZONE = local;
ALTER SESSION SET TIME_ZONE = 'America/New_York';
```

7

The Oracle database supports storing the time zone in your date and time data, as well as fractional seconds.

The `ALTER SESSION` command can be used to change time zone values in a user's session. The time zone values can be set to an absolute offset, a named time zone, a database time zone, or the local time zone. Observe the examples shown in the slide.

Oracle Internal & Oracle Academy Use Only

CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP

- CURRENT_DATE:
  - Returns the current date from the user session
  - Has a data type of DATE
- CURRENT_TIMESTAMP:
  - Returns the current date and time from the user session
  - Has a data type of TIMESTAMP WITH TIME ZONE
- LOCALTIMESTAMP:
  - Returns the current date and time from the user session
  - Has a data type of TIMESTAMP

8

The CURRENT_DATE and CURRENT_TIMESTAMP functions return the current date and current time stamp, respectively. The data type of CURRENT_DATE is DATE. The data type of CURRENT_TIMESTAMP is TIMESTAMP WITH TIME ZONE.

The values returned display the time zone displacement of the SQL session executing the functions. Here, the time zone displacement is the difference (in hours and minutes) between local time and UTC. The TIMESTAMP WITH TIME ZONE data type has the format:

```
TIMESTAMP [ (fractional_seconds_precision) ] WITH TIME ZONE
```

where fractional_seconds_precision optionally specifies the number of digits in the fractional part of the SECOND datetime field and can be a number in the range 0 through 9. The default is 6.

The LOCALTIMESTAMP function returns the current date and time in the session time zone. The difference between LOCALTIMESTAMP and CURRENT_TIMESTAMP is that LOCALTIMESTAMP returns a TIMESTAMP value, whereas CURRENT_TIMESTAMP returns a TIMESTAMP WITH TIME ZONE value.

These functions are national language support (NLS)–sensitive—that is, the results will be in the current NLS calendar and datetime formats.

Note: The SYSDATE function returns the current date and time as a DATE data type. You learned how to use the SYSDATE function in the course titled *Oracle Database: SQL Workshop I*.

Oracle Internal & Oracle Academy Use Only

Comparing Date and Time in a Session's Time Zone

The TIME_ZONE parameter is set to –5:00, and then SELECT statements for each date and time are executed to compare differences.

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
ALTER SESSION SET TIME_ZONE = '-5:00';

SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;                    1

SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL;               2

SELECT SESSIONTIMEZONE, LOCALTIMESTAMP FROM DUAL;                  3
```

9

The ALTER SESSION command sets the date format of the session to 'DD-MON-YYYY HH24:MI:SS' that is:

day of month (1–31) – abbreviated name of month – 4-digit year hour of day (0–23):minute (0–59):second (0–59).

The example in the slide illustrates that the session is altered to set the TIME_ZONE parameter to –5:00. Then the SELECT statement for CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP is executed to observe the differences in format.

Note: The TIME_ZONE parameter specifies the default local time zone displacement for the current SQL session. TIME_ZONE is only a session parameter and not an initialization parameter. The TIME_ZONE parameter is set as follows:

```
TIME_ZONE = '[+ | -] hh:mm'
```

The format mask ([+ | -] hh:mm) indicates the hours and minutes before or after UTC.

Oracle Internal & Oracle Academy Use Only

Comparing Date and Time in a Session's Time Zone

Results of queries:

```
Session altered.
```

1

	SESSIONTIMEZONE	CURRENT_DATE
1	-05:00	28-AUG-2016 18:14:49

2

	SESSIONTIMEZONE	CURRENT_TIMESTAMP
1	-05:00	28-AUG-16 06.15.59.648595000 PM -05:00

3

	SESSIONTIMEZONE	LOCALTIMESTAMP
1	-05:00	28-AUG-16 06.16.59.026930000 PM

10

Observe the results of the queries run from the previous slide:

1. The `CURRENT_DATE` function returns the current date in the session's time zone.
2. The `CURRENT_TIMESTAMP` function returns the current date and time in the session's time zone as a value of the data type `TIMESTAMP WITH TIME ZONE`.
3. The `LOCALTIMESTAMP` function returns the current date and time in the session's time zone.

Note: The code example output may vary depending on when the command is run.

Oracle Internal & Oracle Academy Use Only

DBTIMEZONE and SESSIONTIMEZONE

- Display the value of the database time zone:

```
SELECT DBTIMEZONE FROM DUAL;
```

	DBTIMEZONE
1	+00:00

- Display the value of the session's time zone:

```
SELECT SESSIONTIMEZONE FROM DUAL;
```

	SESSIONTIMEZONE
1	-05:00

11

If you are the DBA, you can set the database's default time zone by specifying the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If you omit, the default database time zone is the operating system time zone. Note that the database time zone cannot be changed for a session with an `ALTER SESSION` statement.

The `DBTIMEZONE` function returns the value of the database 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 database time zone value in the most recent `CREATE DATABASE` or `ALTER DATABASE` statement. The example in the slide shows that the database time zone is set to "`+00:00`," as the `TIME_ZONE` parameter is in the format:

```
TIME_ZONE = '[+ | -] hh:mm'
```

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 offset to `UTC` by `'-5:00'` hours.

Observe that the database time zone is different from the current session's time zone.

Oracle Internal & Oracle Academy Use Only

TIMESTAMP Data Types

Data Type	Fields
TIMESTAMP	Year, Month, Day, Hour, Minute, Second with fractional seconds
TIMESTAMP WITH TIME ZONE	Same as the TIMESTAMP data type; also includes: TIMEZONE_HOUR, and TIMEZONE_MINUTE or TIMEZONE_REGION
TIMESTAMP WITH LOCAL TIME ZONE	Same as the TIMESTAMP data type; also includes a time zone offset in its value

12

The TIMESTAMP data type is an extension of the DATE data type.

```
TIMESTAMP (fractional_seconds_ precision)
```

This data type contains the year, month, and day values of date, as well as hour, minute, and second values of time, where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. The accepted values of significant fractional_seconds_precision are 0 through 9. The default is 6.

```
TIMESTAMP (fractional_seconds_precision) WITH TIME ZONE
```

This data type contains all values of TIMESTAMP as well as time zone displacement value.

```
TIMESTAMP (fractional_seconds_precision) WITH LOCAL TIME ZONE
```

This data type contains all values of TIMESTAMP, with the following exceptions:

- Data is normalized to the database time zone when it is stored in the database.
- When the data is retrieved, users see the data in the session time zone.

Oracle Internal & Oracle Academy Use Only

TIMESTAMP Fields

Datetime Field	Valid Values
YEAR	–4712 to 9999 (excluding year 0)
MONTH	01 to 12
DAY	01 to 31
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 59.9(N) where 9(N) is precision
TIMEZONE_HOUR	–12 to 14
TIMEZONE_MINUTE	00 to 59

13

Each datetime data type is composed of several of these fields. You can mutually compare datetimes and assign them only if they have the same datetime fields.

Oracle Internal & Oracle Academy Use Only

Difference Between DATE and TIMESTAMP

A

```
-- when hire_date is of
type DATE

SELECT hire_date
FROM emp4;
```

	HIRE_DATE
1	17-JUN-11
2	21-SEP-09
3	13-JAN-09
4	03-JAN-14
5	21-MAY-15
6	25-JUN-13
7	05-FEB-14
8	07-FEB-15
9	17-AUG-10
10	16-AUG-10

...

B

```
ALTER TABLE emp4
MODIFY hire_date TIMESTAMP;

SELECT hire_date
FROM emp4;
```

	HIRE_DATE
1	17-JUN-11 12.00.00.000000000 AM
2	21-SEP-09 12.00.00.000000000 AM
3	13-JAN-09 12.00.00.000000000 AM
4	03-JAN-14 12.00.00.000000000 AM
5	21-MAY-15 12.00.00.000000000 AM
6	25-JUN-13 12.00.00.000000000 AM
7	05-FEB-14 12.00.00.000000000 AM
8	07-FEB-15 12.00.00.000000000 AM
9	17-AUG-10 12.00.00.000000000 AM
10	16-AUG-10 12.00.00.000000000 AM

...

14

TIMESTAMP Data Type: Example

Example A shows the data from the hire_date column of the EMP4 table when the data type of the column is DATE.

In example B, the table is altered and the data type of the hire_date column is made into TIMESTAMP. The output shows the differences in display. You can convert from DATE to TIMESTAMP when the column has data, but you cannot convert from DATE or TIMESTAMP to TIMESTAMP WITH TIME ZONE unless the column is blank.

You can specify the fractional seconds precision for time stamp. If none is specified, as in this example, it defaults to 6.

For example, the following statement sets the fractional seconds precision as 7:

```
ALTER TABLE emp4
MODIFY hire_date TIMESTAMP(7);
```

The Oracle DATE data type by default looks like what is shown in this example. However, the date data type also contains additional information such as hours, minutes, seconds, AM, and PM. To obtain the date in this format, you can apply a format mask or a function to the date value.

Note: Remember to change the NLS_DATE_FORMAT to 'DD-MON-YY' before running the queries in the slide.

Oracle Internal & Oracle Academy Use Only

Comparing TIMESTAMP Data Types

```
CREATE TABLE web_orders
       (order_date TIMESTAMP WITH TIME ZONE,
        delivery_time TIMESTAMP WITH LOCAL TIME ZONE);
```

```
INSERT INTO web_orders values
       (current_date, current_timestamp + 2);
```

```
SELECT * FROM web_orders;
```

	ORDER_DATE	DELIVERY_TIME
1	28-AUG-16 06.49.27.000000000 PM -05:00	30-AUG-16 06.49.27.000000000 PM

15

In the example in the slide, a new table `web_orders` is created with a column of data type `TIMESTAMP WITH TIME ZONE` and a column of data type `TIMESTAMP WITH LOCAL TIME ZONE`.

This table is populated whenever a `web_order` is placed. The time stamp and time zone for the user placing the order are inserted based on the `CURRENT_DATE` value. The delivery time is populated by inserting two days from the `CURRENT_TIMESTAMP` value into it every time an order is placed. When a web-based company guarantees shipping, it can estimate its delivery time based on the time zone of the person placing the order.

Note: The code example output may vary based on the time when the command is run.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP
- INTERVAL data types
- Using the following functions:
  - EXTRACT
  - TZ_OFFSET
  - FROM_TZ
  - TO_TIMESTAMP
  - TO_YMINTERVAL
  - TO_DSINTERVAL

16

This section discusses INTERVAL data types.

Oracle Internal & Oracle Academy Use Only

INTERVAL Data Types

- You can use `INTERVAL` data types to store the difference between two datetime values.
- There are two classes of intervals:
  - Year-month
  - Day-time
- The precision of the interval is:
  - The actual subset of fields that constitutes an interval
  - Specified in the interval qualifier

Data Type	Fields
INTERVAL YEAR TO MONTH	Year, Month
INTERVAL DAY TO SECOND	Days, Hour, Minute, Second with fractional seconds

17

`INTERVAL` data types are used to store the difference between two datetime values. There are two classes of intervals: year-month intervals and day-time intervals.

A year-month interval is made up of a contiguous subset of fields of `YEAR` and `MONTH`, whereas a day-time interval is made up of a contiguous subset of fields consisting of `DAY`, `HOUR`, `MINUTE`, and `SECOND`. The actual subset of fields that constitute an interval is called the precision of the interval and is specified in the interval qualifier. Because the number of days in a year is calendar-dependent, the year-month interval is NLS-dependent, whereas day-time interval is NLS-independent.

The interval qualifier contains a leading field and may contain a trailing field. In case the trailing field is `SECOND`, it may also specify the fractional seconds precision, which is the number of digits in the fractional part of the `SECOND` value. If not specified, the default value for leading field precision is 2 digits and the default value for fractional seconds precision is 6 digits.

`INTERVAL YEAR (year_precision) TO MONTH`

This data type stores a period of time in years and months, where `year_precision` is the number of digits in the `YEAR` datetime field. The accepted values are 0 through 9. The default is 6.

`INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision)`

This data type stores a period of time in days, hours, minutes, and seconds, where `day_precision` is the maximum number of digits in the `DAY` datetime field (accepted values are 0 through 9; the default is 2) and `fractional_seconds_precision` is the number of digits in the fractional part of the `SECOND` field (accepted values are 0 through 9; the default is 6).

Oracle Internal & Oracle Academy Use Only

INTERVAL Fields

INTERVAL Field	Valid Values for Interval
YEAR	Any positive or negative integer
MONTH	00 to 11
DAY	Any positive or negative integer
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 59.9(N) where 9(N) is precision

18

INTERVAL YEAR TO MONTH can have fields of YEAR and MONTH.

INTERVAL DAY TO SECOND can have fields of DAY, HOUR, MINUTE, and SECOND.

The actual subset of fields that constitute an item of either type of interval is defined by an interval qualifier, and this subset is known as the precision of the item.

Year-month intervals are mutually comparable and assignable only with other year-month intervals. Day-time intervals are mutually comparable and assignable only with other day-time intervals.

Oracle Internal & Oracle Academy Use Only

INTERVAL YEAR TO MONTH: Example

```
CREATE TABLE warranty
(prod_id number,  warranty_time INTERVAL YEAR(3) TO MONTH);
INSERT INTO warranty VALUES (123, INTERVAL '8' MONTH);
INSERT INTO warranty VALUES (155, INTERVAL '200' YEAR(3));
INSERT INTO warranty VALUES (678, '200-11');
SELECT * FROM warranty;
```

	PROD_ID	WARRANTY_TIME
1	123	0-8
2	155	200-0
3	678	200-11

19

Oracle Internal & Oracle Academy Use Only

`INTERVAL YEAR TO MONTH` stores a period of time by using the `YEAR` and `MONTH` datetime fields. Specify `INTERVAL YEAR TO MONTH` as follows:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

where `year_precision` is the number of digits in the `YEAR` datetime field. The default value of `year_precision` is 2.

Restriction: The leading field must be more significant than the trailing field. For example, `INTERVAL '0-1' MONTH TO YEAR` is not valid.

Examples:

- `INTERVAL '123-2' YEAR(3) TO MONTH`

  Indicates an interval of 123 years, 2 months
- `INTERVAL '123' YEAR(3)`

  Indicates an interval of 123 years, 0 months
- `INTERVAL '300' MONTH(3)`

  Indicates an interval of 300 months
- `INTERVAL '123' YEAR`

  Returns an error because the default precision is 2, and '123' has 3

The Oracle database supports two interval data types: `INTERVAL YEAR TO MONTH` and `INTERVAL DAY TO SECOND`. For interval literals, the system also recognizes other American National Standards Institute (ANSI) interval types, such as `INTERVAL '2' YEAR` or `INTERVAL '10' HOUR`. In these cases, each interval is converted to one of the two supported types.

In the example in the slide, a `WARRANTY` table is created, which contains a `warranty_time` column that takes the `INTERVAL YEAR(3) TO MONTH` data type. Different values are inserted into it to indicate years and months for various products. When these rows are retrieved from the table, you see a year value separated from the month value by a (–).

Oracle Internal & Oracle Academy Use Only

INTERVAL DAY TO SECOND Data Type: Example

```
CREATE TABLE lab
( exp_id number, test_time INTERVAL DAY(2) TO SECOND);

INSERT INTO lab VALUES (100012, '90 00:00:00');
INSERT INTO lab VALUES (56098,
  INTERVAL '6 03:30:16' DAY    TO SECOND);
```

```
SELECT * FROM lab;
```

	EXP_ID	TEST_TIME
1	100012	90 0:0:0.0
2	56098	6 3:30:16.0

21

In the example in the slide:

- You create the `lab` table with a `test_time` column of the `INTERVAL DAY TO SECOND` data type.
- You then insert into it the value `'90 00:00:00'` to indicate 90 days and 0 hours, 0 minutes, and 0 seconds.
- You insert another row with the value `INTERVAL '6 03:30:16' DAY TO SECOND` to indicate 6 days, 3 hours, 30 minutes, and 16 seconds.
- The `SELECT` statement shows how this data is displayed in the database.

Oracle Internal & Oracle Academy Use Only

Lesson Agenda

- CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP
- INTERVAL data types
- Using the following functions:
  - EXTRACT
  - TZ_OFFSET
  - FROM_TZ
  - TO_TIMESTAMP
  - TO_YMINTERVAL
  - TO_DSINTERVAL

22

This section discusses the following functions:

- EXTRACT
- TZ_OFFSET
- FROM_TZ
- TO_TIMESTAMP
- TO_YMINTERVAL
- TO_DSINTERVAL

Oracle Internal & Oracle Academy Use Only

EXTRACT

- Display all employees who were hired after 2007.

```
SELECT last_name, employee_id, hire_date
  FROM employees
  WHERE EXTRACT(YEAR FROM TO_DATE(hire_date, 'DD-MON-RR')) > 2007
  ORDER BY hire_date;
```

- Display the MONTH component from the HIRE_DATE for those employees whose MANAGER_ID is 100.

```
SELECT last_name, hire_date,
       EXTRACT (MONTH FROM HIRE_DATE)
FROM employees
WHERE manager_id = 100;
```

	LAST_NAME	HIRE_DATE	EXTRACT(MONTHFROMHIRE_DATE)
1	Kochhar	21-SEP-09	9
2	De Haan	13-JAN-09	1
3	Raphaely	07-DEC-10	12
4	Weiss	18-JUL-12	7
5	Fripp	10-APR-13	4

...

23

The EXTRACT expression extracts and returns the value of a specified datetime field from a datetime or interval value expression. You can extract any of the components mentioned in the following syntax by using the EXTRACT function. The syntax of the EXTRACT function is:

```
SELECT EXTRACT( { YEAR | MONTH | DAY | HOUR | MINUTE | SECOND
 | TIMEZONE_HOUR
 | TIMEZONE_MINUTE
 | TIMEZONE_REGION
 | TIMEZONE_ABBR }
 FROM { expr } )
```

When you extract a TIMEZONE_REGION or TIMEZONE_ABBR (abbreviation), the value returned is a string containing the appropriate time zone name or abbreviation. When you extract any of the other values, the value returned is a date in the Gregorian calendar. When extracting from a datetime with a time zone value, the value returned is in UTC.

In the first example in the slide, the EXTRACT function is used to select all employees who were hired after 2007.

In the second example, the EXTRACT function is used to extract the MONTH from the HIRE_DATE column of the EMPLOYEES table for those employees who report to the manager whose EMPLOYEE_ID is 100.

Oracle Internal & Oracle Academy Use Only

TZ_OFFSET

Display the time zone offset for the 'US/Eastern', 'Canada/Yukon', and 'Europe/London' time zones:

```
SELECT TZ_OFFSET('US/Eastern'),
       TZ_OFFSET('Canada/Yukon'),
       TZ_OFFSET('Europe/London')
 FROM DUAL;
```

	TZ_OFFSET('US/EASTERN')	TZ_OFFSET('CANADA/YUKON')	TZ_OFFSET('EUROPE/LONDON')
1	-04:00	-07:00	+01:00

24

The TZ_OFFSET function returns the time zone offset corresponding to the value entered. The return value is dependent on the time when the statement is executed. For example, if the TZ_OFFSET function returns a value –08:00, this value indicates that the time zone for which the command was executed is eight hours behind UTC. You can enter a valid time zone name, a time zone offset from UTC (which simply returns itself), or the keyword SESSIONTIMEZONE or DBTIMEZONE. The syntax of the TZ_OFFSET function is:

```
TZ_OFFSET({ 'time_zone_name'| '{ + | - } hh : mi'
        | SESSIONTIMEZONE
        | DBTMEZONE
        })
```

The Fold Motor Company has its headquarters in Michigan, USA, which is in the US/Eastern time zone. The company president, Mr. Fold, wants to conduct a conference call with the vice president of the Canadian operations and the vice president of the European operations, who are in the Canada/Yukon and Europe/London time zones, respectively. Mr. Fold wants to find out the time in each of these places to make sure that his senior management will be available to attend the meeting. His secretary, Mr. Scott, helps by issuing the queries shown in the example and gets the following results:

- The 'US/Eastern' time zone is four hours behind UTC.
- The 'Canada/Yukon' time zone is seven hours behind UTC.
- The 'Europe/London' time zone is one hour ahead of UTC.

Oracle Internal & Oracle Academy Use Only

For a listing of valid time zone name values, you can query the `V$TIMEZONE_NAMES` dynamic performance view.

```
SELECT * FROM V$TIMEZONE_NAMES;
```

	TZNAME	TZABBREV	CON_ID
1	Africa/Abidjan	LMT	0
2	Africa/Abidjan	GMT	0
3	Africa/Accra	LMT	0
4	Africa/Accra	GMT	0
5	Africa/Accra	GHST	0

...

Oracle Internal & Oracle Academy Use Only

FROM_TZ

Display the TIMESTAMP value '2000-07-12 08:00:00' as a TIMESTAMP WITH TIME ZONE value for the 'Australia/North' time zone region.

```
SELECT FROM_TZ(TIMESTAMP
       '2000-07-12 08:00:00', 'Australia/North')
FROM DUAL;
```

	FROM_TZ(TIMESTAMP'2000-07-1208:00:00','AUSTRALIA/NORTH')
1	12-JUL-00 08.00.00.000000000 AM AUSTRALIA/NORTH

26

The FROM_TZ function converts a TIMESTAMP value to a TIMESTAMP WITH TIME ZONE value.

The syntax of the FROM_TZ function is as follows:

FROM_TZ(timestamp_value, time_zone_value)

where time_zone_value is a character string in the format 'TZH:TZM' or a character expression that returns a string in TZR (time zone region) with an optional TZD format. TZD is an abbreviated time zone string with daylight saving information. TZR represents the time zone region in datetime input strings. Examples are 'Australia/North', 'PST' for US/Pacific standard time, 'PDT' for US/Pacific daylight time, and so on.

The example in the slide converts a TIMESTAMP value to TIMESTAMP WITH TIME ZONE.

Note: To see a listing of valid values for the TZR and TZD format elements, query the V$TIMEZONE_NAMES dynamic performance view.

Oracle Internal & Oracle Academy Use Only

TO_TIMESTAMP

Display the character string '2016-03-06 11:00:00' as a TIMESTAMP value:

```
SELECT TO_TIMESTAMP ('2016-03-06 11:00:00',
                     'YYYY-MM-DD HH:MI:SS')
FROM DUAL;
```

	TO_TIMESTAMP('2016-03-0611:00:00','YYYY-MM-DDHH:MI:SS')
1	06-MAR-16 11.00.00.000000000 AM

27

The TO_TIMESTAMP function converts a string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of the TIMESTAMP data type. The syntax of the TO_TIMESTAMP function is:

```
TO_TIMESTAMP(char [, fmt [, 'nlsparam' ] ])
```

The optional fmt specifies the format of char. If you omit fmt, the string must be in the default format of the TIMESTAMP data type. The optional nlsparam specifies the language in which month and day names, and abbreviations, are returned. This argument can have this form:

```
'NLS_DATE_LANGUAGE = language'
```

If you omit nlsparams, this function uses the default date language for your session.

The example in the slide converts a character string to a value of TIMESTAMP.

Note: You use the TO_TIMESTAMP_TZ function to convert a string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of the TIMESTAMP WITH TIME ZONE data type. For more information about this function, see *Oracle Database SQL Language Reference* for Oracle Database 19c.

Oracle Internal & Oracle Academy Use Only

TO_YMINTERVAL

Display a date that is one year and two months after the hire date for the employees working in the department with the DEPARTMENT_ID 20.

```
SELECT hire_date,
       hire_date + TO_YMINTERVAL('01-02') AS
       HIRE_DATE_YMININTERVAL
FROM   employees
WHERE department_id = 20;
```

	HIRE_DATE	HIRE_DATE_YMINTERVAL
1	17-FEB-12	17-APR-13
2	17-AUG-13	17-OCT-14

28

The TO_YMINTERVAL function converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to an INTERVAL YEAR TO MONTH data type. The INTERVAL YEAR TO MONTH data type stores a period of time using the YEAR and MONTH datetime fields. The format of INTERVAL YEAR TO MONTH is as follows:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

where year_precision is the number of digits in the YEAR datetime field. The default value of year_precision is 2.
The syntax of the TO_YMINTERVAL function is:

```
TO_YMINTERVAL (char)
```

where char is the character string to be converted.
The example in the slide calculates a date that is one year and two months after the hire date for the employees working in the department with the DEPARTMENT_ID 20 of the EMPLOYEES table.

Oracle Internal & Oracle Academy Use Only

TO_DSINTERVAL

Display a date that is 100 days and 10 hours after the hire date for all the employees.

```
SELECT last_name,
 TO_CHAR(hire_date, 'mm-dd-yy:hh:mi:ss') hire_date,
  TO_CHAR(hire_date +
   TO_DSINTERVAL('100 10:00:00'),
     'mm-dd-yy:hh:mi:ss') hiredate2
FROM employees;
```

	LAST_NAME	HIRE_DATE	HIREDATE2
1	King	06-17-11:12:00:00	09-25-11:10:00:00
2	Kochhar	09-21-09:12:00:00	12-30-09:10:00:00
3	De Haan	01-13-09:12:00:00	04-23-09:10:00:00
4	Hunold	01-03-14:12:00:00	04-13-14:10:00:00
5	Ernst	05-21-15:12:00:00	08-29-15:10:00:00
6	Austin	06-25-13:12:00:00	10-03-13:10:00:00

...

29

TO_DSINTERVAL converts a character string of the CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to an INTERVAL DAY TO SECOND data type.

In the example in the slide, the date 100 days and 10 hours after the hire date is obtained.

Oracle Internal & Oracle Academy Use Only

Daylight Saving Time (DST)

- Start of Daylight Saving:
  - Time jumps from 01:59:59 AM to 03:00:00 AM.
  - Values from 02:00:00 AM to 02:59:59 AM are not valid.
- End of Daylight Saving:
  - Time jumps back from 02:00:00 AM to 01:00:01 AM.
  - Values from 01:00:01 AM to 02:00:00 AM are ambiguous because they are visited twice.

30

Most Western nations advance the clock ahead one hour during the summer months. This period is called daylight saving time (DST). The DST lasts from the start of Daylight Saving to the end of Daylight Saving in most of the US, Mexico, and Canada. The nations of the European Union observe DST, but they call it the summertime period. Europe's summertime period begins a week earlier than its North American counterpart but ends at the same time.

The Oracle database automatically determines, for any given time zone region, whether the DST is in effect and returns local time values accordingly. The datetime value is sufficient for the Oracle database to determine whether daylight saving time is in effect for a given region in all cases except boundary cases.

A boundary case occurs during the period when the DST goes into or out of effect. For example, in the US/Eastern region, when DST goes into effect, the time changes from 01:59:59 AM to 03:00:00 AM. The one-hour interval between 02:00:00 AM and 02:59:59 AM. does not exist. When daylight saving time goes out of effect, the time changes from 02:00:00 AM back to 01:00:01 AM, and the one-hour interval between 01:00:01 AM and 02:00:00 AM is repeated.

ERROR_ON_OVERLAP_TIME

The `ERROR_ON_OVERLAP_TIME` is a session parameter to notify the system to issue an error when it encounters a datetime that occurs in the overlapped period and no time zone abbreviation was specified to distinguish the period.

For example, the DST ends on October 31, at 02:00:01 AM. The overlapped periods are:

- 10/31/2016 01:00:01 AM to 10/31/2016 02:00:00 AM (EDT)
- 10/31/2016 01:00:01 AM to 10/31/2016 02:00:00 AM (EST)

Oracle Internal & Oracle Academy Use Only

If you input a datetime string that occurs in one of these two periods, you need to specify the time zone abbreviation (for example, EDT or EST) in the input string for the system to determine the period.

Without this time zone abbreviation, the system does the following:

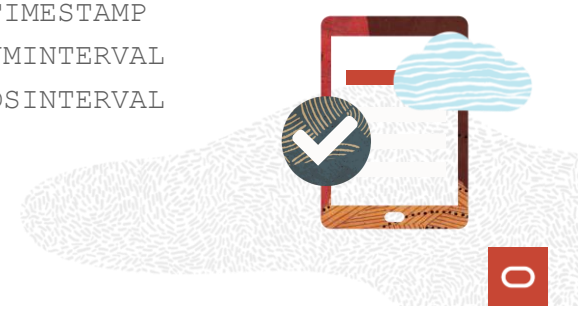
If the `ERROR_ON_OVERLAP_TIME` parameter is `FALSE`, it assumes that the input time is standard time (for example, EST). Otherwise, an error is raised

Oracle Internal & Oracle Academy Use Only

Summary

In this lesson, you should have learned how to use:

- Data types similar to `DATE` that store fractional seconds and track time zones
- Data types that store the difference between two datetime values
- The following datetime functions:
  - `CURRENT_DATE`
  - `CURRENT_TIMESTAMP`
  - `LOCALTIMESTAMP`
  - `DBTIMEZONE`
  - `SESSIONTIMEZONE`
  - `EXTRACT`
  - `TZ_OFFSET`
  - `FROM_TZ`
  - `TO_TIMESTAMP`
  - `TO_YMINTERVAL`
  - `TO_DSINTERVAL`



32

This lesson addressed some of the datetime functions available in the Oracle Database.

Oracle Internal & Oracle Academy Use Only

Practice 20: Overview

This practice focuses on using the datetime functions.

33

In this practice, you display time zone offsets, `CURRENT_DATE`, `CURRENT_TIMESTAMP`, and `LOCALTIMESTAMP`. You also set time zones and use the `EXTRACT` function.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

21

Conclusion

Oracle Internal & Oracle Academy Use Only

Course Goals

After completing this course, you should be able to:

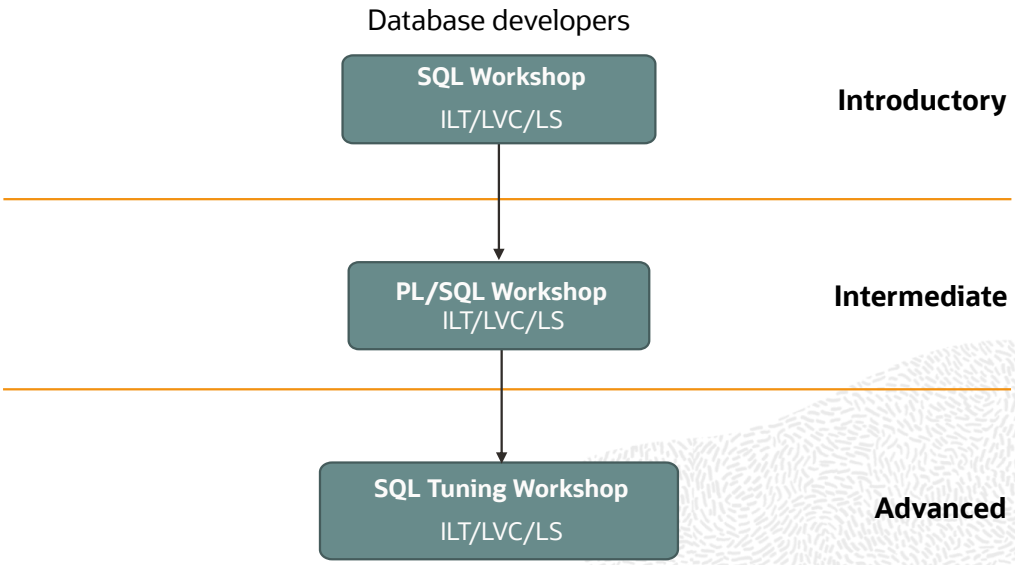
- Identify the major components of an SQL database
- Retrieve row and column data from tables with the SELECT statement
- Create reports of sorted and restricted data
- Employ SQL functions to generate and retrieve customized data
- Run complex queries to retrieve data from multiple tables
- Run data manipulation language (DML) statements to update data in a database
- Run data definition language (DDL) statements to create and manage schema objects

2

This course offers you an introduction to 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

Oracle University: Oracle SQL Training



3

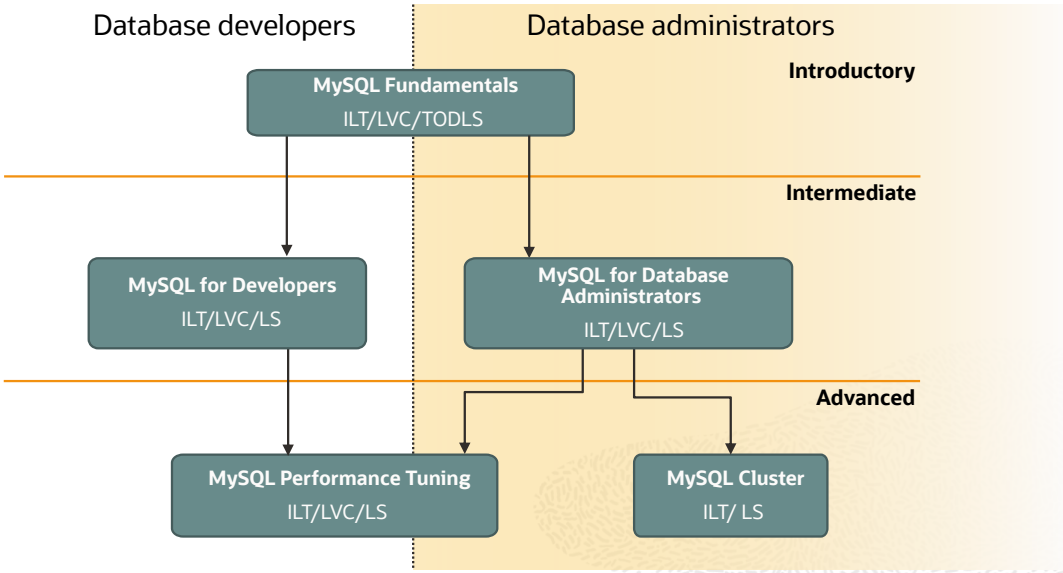
Training Formats

- **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
- **Learning Subscription (LS):** Oracle Cloud Learning Subscriptions provide unlimited access for 12 months to a rich learning ecosystem with constantly changing content that enables you to build expertise and validate your expertise by achieving an oracle certification.

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

Oracle Internal & Oracle Academy Use Only

Oracle University: MySQL Training



4

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
- **Learning Subscription (LS):** Oracle Cloud Learning Subscriptions provide unlimited access for 12 months to a rich learning ecosystem with constantly changing content that enables you to build expertise and validate your expertise by achieving an oracle certification.

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

Oracle Internal & Oracle Academy Use Only

Oracle SQL References

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

- *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

5

Oracle Internal & Oracle Academy Use Only

MySQL Websites

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

6

You can download trial versions of MySQL Enterprise Edition software, view detailed product information, and find out more about Oracle MySQL services at http://www.mysql.com. You can 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.

Oracle Internal & Oracle Academy Use Only

Your Evaluation

- Courses are continually updated, so your feedback is invaluable.
- Thank you for taking the time to give your opinions.

7

Oracle Internal & Oracle Academy Use Only

Thank You

- Congratulations on completing this course!
- Your attendance and participation are appreciated.
- For training and contact information, see the Oracle University website at http://www.oracle.com/education.

8

Oracle Internal & Oracle Academy Use Only

Q&A Session

- Questions and answers
- Questions after class
  - Get answers about Oracle SQL from https://www.oracle.com/database/technologies/appdev/sql.html.
  - Get answers about MySQL from the online reference manual at http://dev.mysql.com/doc/mysql/en/faqs.html.

9

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

A

Table Descriptions

Oracle Internal & Oracle Academy Use Only

Schema Description

Overall Description

The Oracle Database sample schemas portray a sample company that operates worldwide to fill orders for several different products. The company has three divisions:

- **Human Resources:** Tracks information about employees and facilities
- **Order Entry:** Tracks product inventories and sales through various channels
- **Sales History:** Tracks business statistics to facilitate business decisions

Each of these divisions is represented by a schema. In this course, you have access to the objects in all the schemas. However, the emphasis of the examples, demonstrations, and practices is on the `Human Resources` (`HR`) schema.

All scripts necessary to create the sample schemas reside in the `$ORACLE_HOME/demo/schema/` folder.

Human Resources (`HR`)

This is the schema that is used in this course. In the Human Resource (`HR`) records, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary.

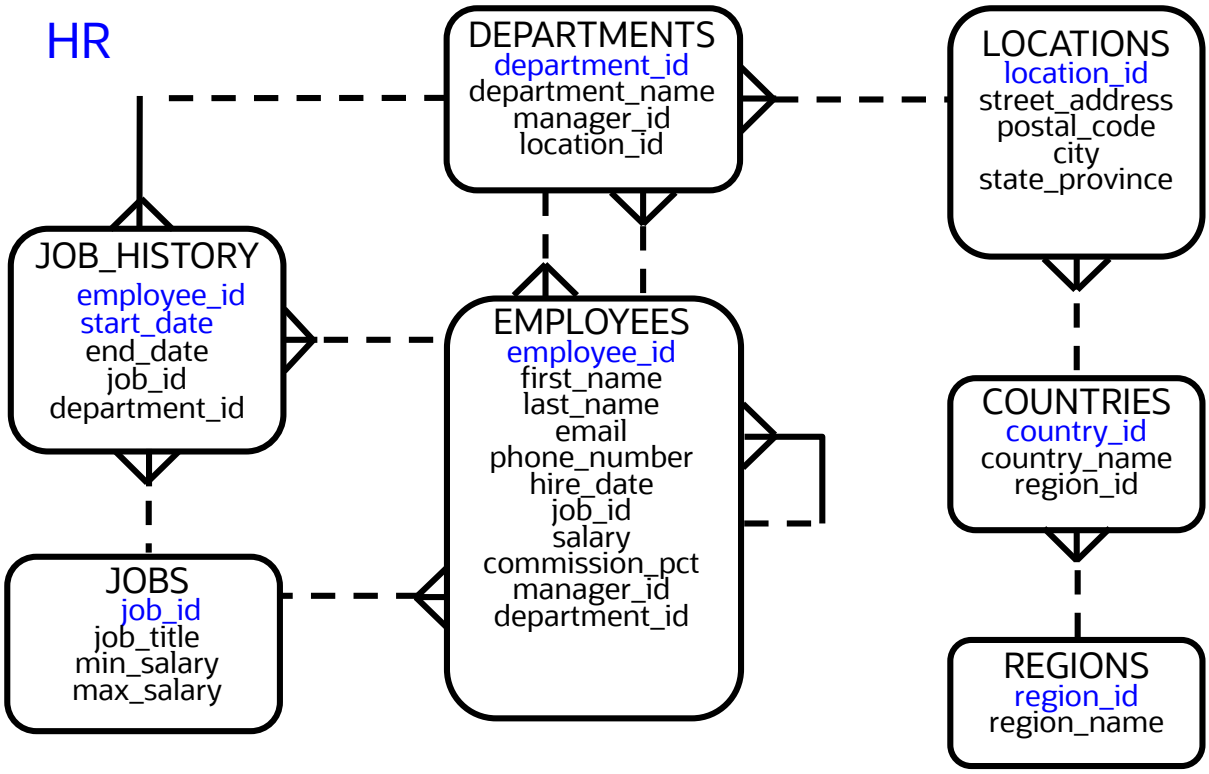
The company also tracks information about the jobs within the organization. Each job has an identification code, job title, and a minimum and maximum salary range for the job. Some employees have been with the company for a long time and have held different positions within the company. When an employee resigns, the duration the employee was working for, the job identification number, and the department are recorded.

The sample company is regionally diverse, so it tracks the locations of its warehouses and departments. Each employee is assigned to a department, and each department is identified either by a unique department number or a short name. Each department is associated with one location, and each location has a full address that includes the street name, postal code, city, state or province, and the country code.

In places where the departments and warehouses are located, the company records details such as the country name, currency symbol, currency name, and the region where the country is located geographically.

Oracle Internal & Oracle Academy Use Only

HR Entity Relationship Diagram



Oracle Internal & Oracle Academy Use Only

Human Resources (HR) Table Descriptions

```
DESCRIBE countries
```

```
Name                          Null     Type
----------------------------- -------- --------------
COUNTRY_ID                    NOT NULL CHAR(2)
COUNTRY_NAME                           VARCHAR2(40)
REGION_ID                              NUMBER
```

```
SELECT * FROM countries
```

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	CA	Canada	2
2	DE	Germany	1
3	UK	United Kingdom	1
4	US	United States of America	2

Oracle Internal & Oracle Academy Use Only

```
DESCRIBE departments
```

```
Name                          Null     Type
----------------------------- -------- --------------
DEPARTMENT_ID                 NOT NULL NUMBER(4)
DEPARTMENT_NAME               NOT NULL VARCHAR2(30)
MANAGER_ID                             NUMBER(6)
LOCATION_ID                            NUMBER(4)
```

```
SELECT * FROM departments
```

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

Oracle Internal & Oracle Academy Use Only

```
DESCRIBE employees
```

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

```
SELECT * FROM employees
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-11	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-09	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-09	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-14	AC_MGR	12008	(null)	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-15	IT_PROG	6000	(null)	103	60
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-15	IT_PROG	4200	(null)	103	60
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-15	ST_MAN	5800	(null)	100	50
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-11	ST_CLERK	3500	(null)	124	50
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-13	ST_CLERK	3100	(null)	124	50
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-14	ST_CLERK	2600	(null)	124	50
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-14	ST_CLERK	2500	(null)	124	50
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-16	SA_MAN	10500	0.2	100	80
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-12	SA_REP	11000	0.3	149	80
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-14	SA_REP	8600	0.2	149	80
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-15	SA_REP	7000	0.15	149	(null)
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-11	AD_ASST	4400	(null)	101	10
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-12	MK_MAN	13000	(null)	100	20
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-13	MK_REP	6000	(null)	201	20
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-10	AC_MGR	12008	(null)	101	110
20	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-10	AC_ACCOUNT	8300	(null)	205	80

Oracle Internal & Oracle Academy Use Only

```
DESCRIBE job_history
```

```
Name                          Null     Type
----------------------------- -------- --------------
EMPLOYEE_ID                   NOT NULL NUMBER(6)
START_DATE                    NOT NULL DATE
END_DATE                      NOT NULL DATE
JOB_ID                        NOT NULL VARCHAR2(10)
DEPARTMENT_ID                          NUMBER(4)
```

```
SELECT * FROM job_history
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-09	24-JUL-14	IT_PROG	60
2	101	21-SEP-09	27-OCT-16	AC_ACCOUNT	110
3	101	28-OCT-09	15-MAR-13	AC_MGR	110
4	201	17-FEB-12	19-DEC-15	MK_REP	20
5	114	24-MAR-14	31-DEC-15	ST_CLERK	50
6	122	01-JAN-15	31-DEC-15	ST_CLERK	50
7	200	17-SEP-95	17-JUN-09	AD_ASST	90
8	176	24-MAR-14	31-DEC-14	SA_REP	80
9	176	01-JAN-15	31-DEC-15	SA_MAN	80
10	200	01-JUL-10	31-DEC-14	AC_ACCOUNT	90

Oracle Internal & Oracle Academy Use Only

```
DESCRIBE jobs
```

```
Name                            Null     Type
------------------------------- -------- --------------
JOB_ID                          NOT NULL VARCHAR2(10)
JOB_TITLE                       NOT NULL VARCHAR2(35)
MIN_SALARY                               NUMBER(6)
MAX_SALARY                               NUMBER(6)
```

```
SELECT * FROM jobs
```

	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES	President	20080	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	AC_MGR	Accounting Manager	8200	16000
5	AC_ACCOUNT	Public Accountant	4200	9000
6	SA_MAN	Sales Manager	10000	20080
7	SA_REP	Sales Representative	6000	12008
8	ST_MAN	Stock Manager	5500	8500
9	ST_CLERK	Stock Clerk	2008	5000
10	IT_PROG	Programmer	4000	10000
11	MK_MAN	Marketing Manager	9000	15000
12	MK_REP	Marketing Representative	4000	9000

Oracle Internal & Oracle Academy Use Only

```
DESCRIBE locations
```

```
Name                           Null     Type
------------------------------ -------- ---------------
LOCATION_ID                    NOT NULL NUMBER(4)
STREET_ADDRESS                          VARCHAR2(40)
POSTAL_CODE                             VARCHAR2(12)
CITY                           NOT NULL VARCHAR2(30)
STATE_PROVINCE                          VARCHAR2(25)
COUNTRY_ID                              CHAR(2)
```

```
SELECT * FROM locations
```

	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	1500	2011 Interiors Blvd	99236	South San Francisco	California	US
3	1700	2004 Charade Rd	98199	Seattle	Washington	US
4	1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

Oracle Internal & Oracle Academy Use Only

```
DESCRIBE regions
```

```
Name                           Null     Type
------------------------------ -------- --------------
REGION_ID                      NOT NULL NUMBER
REGION_NAME                             VARCHAR2(25)
```

```
SELECT * FROM regions
```

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa

Oracle Internal & Oracle Academy Use Only

ORACLE

B

Using SQL Developer

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports
- Browse the data modeling options in SQL Developer

2

In this appendix, you are introduced to the graphical tool called SQL Developer. You learn how to use SQL Developer for your database development tasks. You learn how to use SQL Worksheet to execute SQL statements and SQL scripts.

Oracle Internal & Oracle Academy Use Only

What Is Oracle SQL Developer?

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

3

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

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

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

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

SQL Developer is the interface to administer the Oracle Application Express Listener. The new interface enables you to specify global settings and multiple database settings with different database connections for the Application Express Listener. SQL Developer provides the option to drag and drop objects by table or column name onto the worksheet. It provides improved DB Diff comparison options, `GRANT` statement support in the SQL editor, and DB Doc reporting. Additionally, SQL Developer includes support for Oracle Database 19c features.

Oracle Internal & Oracle Academy Use Only

Specifications of SQL Developer

SQL Developer:

- Is shipped along with Oracle Database 19c Release 2
- Is developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity using the JDBC Thin driver
- Connects to Oracle Database version 9.2.0.1 and later

4

Oracle SQL Developer is shipped along with Oracle Database 19c by default. SQL Developer is developed in Java, leveraging the Oracle JDeveloper integrated development environment (IDE). Therefore, it is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms.

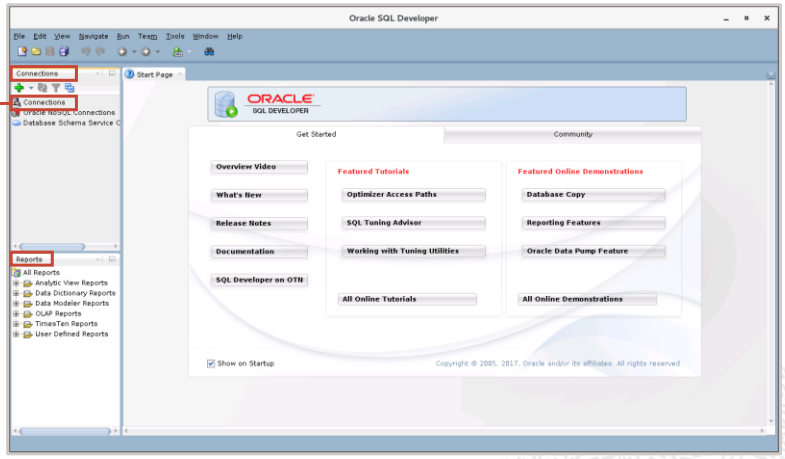
The default connectivity to the database is through the Java Database Connectivity (JDBC) Thin driver, and therefore, no Oracle Home is required. SQL Developer does not require an installer, and you need to simply unzip the downloaded file. With SQL Developer, users can connect to Oracle Databases 9.2.0.1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

Note: For the latest version of SQL Developer, you will have to download and install SQL Developer. SQL Developer is freely downloadable from the following link:
http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html

For instructions on how to install SQL Developer, see the website at:
http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html

Oracle Internal & Oracle Academy Use Only

SQL Developer 17.4.1 Interface



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

5

The SQL Developer interface contains two main navigation tabs:

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

General Navigation and Use

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

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

Menus

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

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options

Oracle Internal & Oracle Academy Use Only

- **Versioning:** Provides integrated support for the following versioning and source control systems—Concurrent Versions System (CVS) and Subversion
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet. It also contains options related to migrating third-party databases to Oracle.

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

Oracle Internal & Oracle Academy Use Only

Creating a Database Connection

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

7

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

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

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

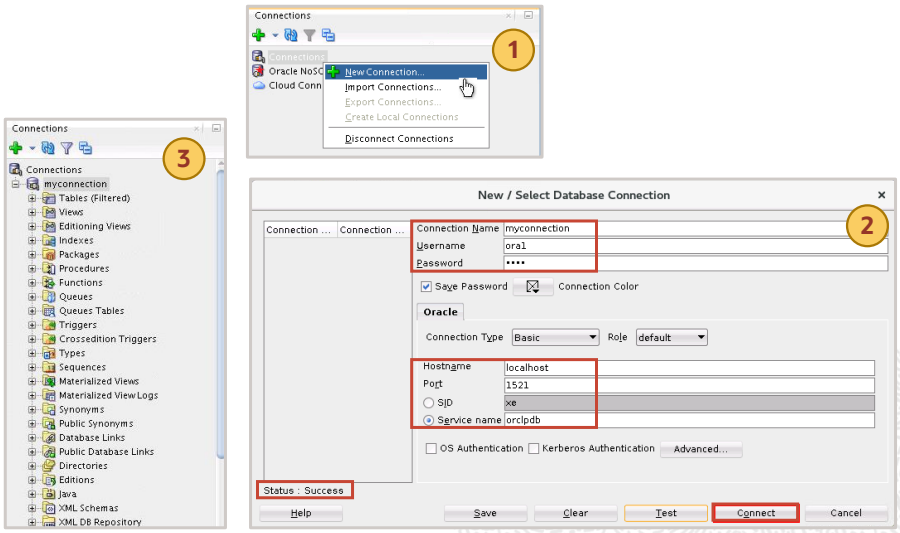
Note: On Windows, if the `tnsnames.ora` file exists, but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

You can export connections to an Extensible Markup Language (XML) file so that you can reuse it.

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

Oracle Internal & Oracle Academy Use Only

Creating a Database Connection



8

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click Connections and select New Connection.
2. In the New / Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
   a. From the Role drop-down list, you can select either *default* or `SYSDBA`. (You choose `SYSDBA` for the `sys` user or any user with database administrator privileges.)
   b. You can select the connection type as:

      Basic: In this type, enter host name and SID for the database that you want to connect to. Port is already set to 1521. You can also choose to enter the Service name directly if you use a remote database connection.

      TNS: You can select any one of the database aliases imported from the `tnsnames.ora` file.

      LDAP: You can look up database services in Oracle Internet Directory, which is a component of Oracle Identity Management.

      Advanced: You can define a custom Java Database Connectivity (JDBC) URL to connect to the database.

      Local/Bequeath: If the client and database exist on the same computer, a client connection can be passed directly to a dedicated server process without going through the listener.
   c. Click Test to ensure that the connection has been set correctly.
   d. Click Connect.

Oracle Internal & Oracle Academy Use Only

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

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

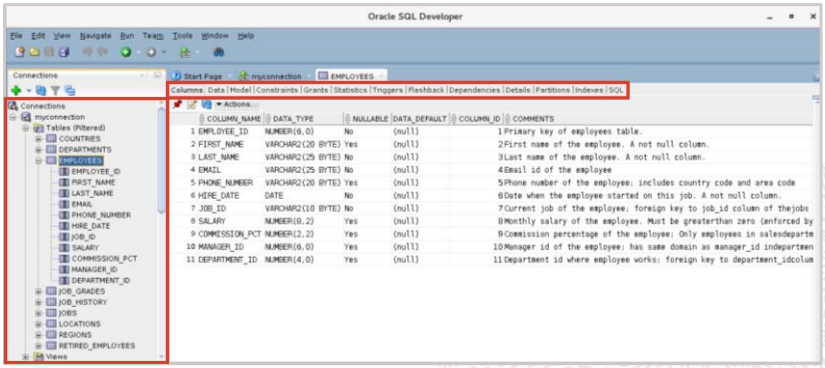
Note: From the same New/Select Database Connection window, you can define connections to non-Oracle data sources using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Oracle Internal & Oracle Academy Use Only

Browsing Database Objects

Use the Connections Navigator to:

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



10

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

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

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

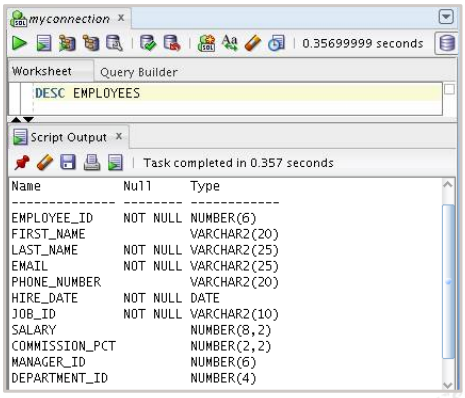
If you want to see the definition of the `EMPLOYEES` table as shown in the slide, perform the following steps:

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

Oracle Internal & Oracle Academy Use Only

Displaying the Table Structure

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



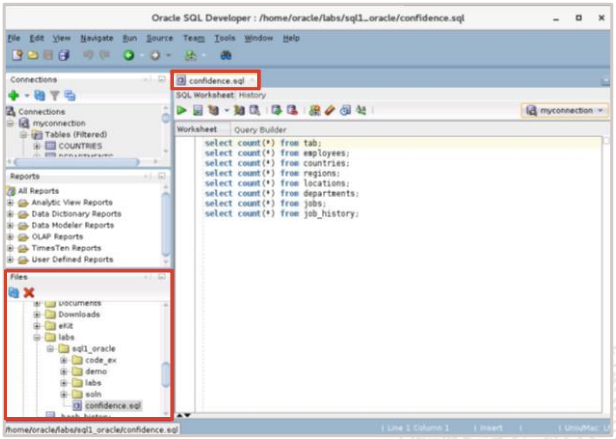
11

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

Oracle Internal & Oracle Academy Use Only

Browsing Files

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



12

Browsing Database Objects

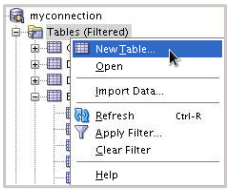
You can use the File Navigator to browse and open system files.

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

Oracle Internal & Oracle Academy Use Only

Creating a Schema Object

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



13

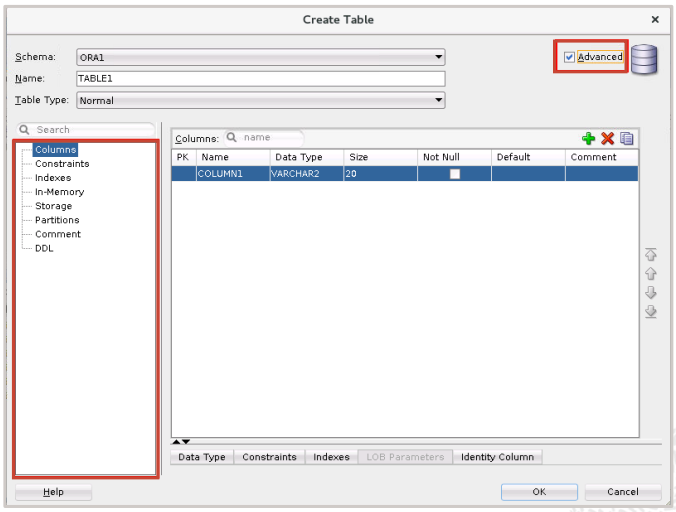
SQL Developer supports the creation of any schema object by executing a SQL statement in SQL Worksheet. Alternatively, you can create objects by using the context menus. When created, you can edit objects using an edit dialog box or one of the many context-sensitive menus.

As new objects are created or existing objects are edited, the DDL for those adjustments is available for review. An Export DDL option is available if you want to create the full DDL for one or more objects in the schema.

The slide shows how to create a table using the context menu. To open a dialog box for creating a new table, right-click Tables and select New Table. The dialog boxes to create and edit database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Oracle Internal & Oracle Academy Use Only

Creating a New Table: Example



14

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

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

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

To create a new table, perform the following steps:

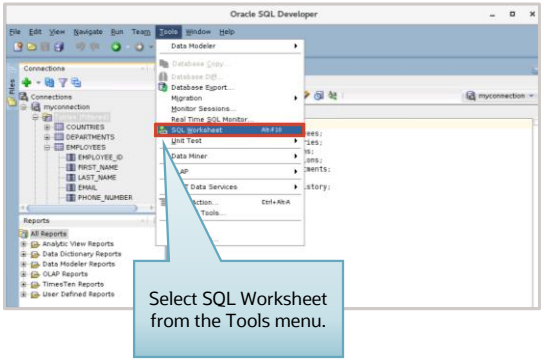
1. In the Connections Navigator, right-click Tables and select Create TABLE.
2. In the Create Table dialog box, select Advanced.
3. Specify the column information.
4. Click OK.

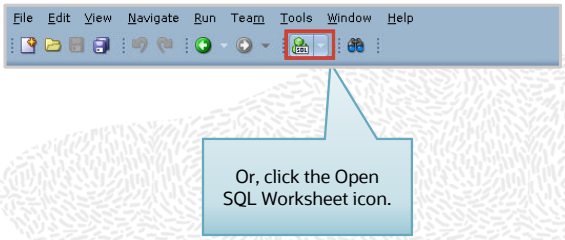
Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

Oracle Internal & Oracle Academy Use Only

Using the SQL Worksheet

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





15

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

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

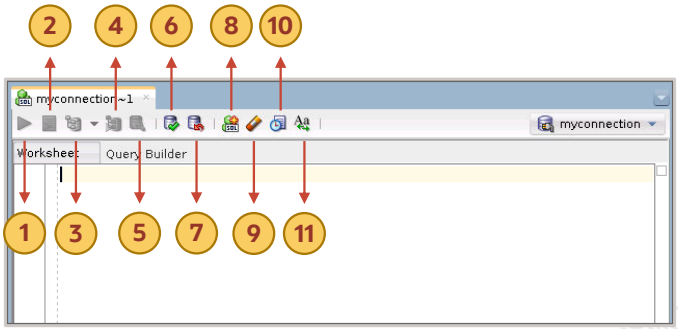
- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

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

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

Oracle Internal & Oracle Academy Use Only

Using the SQL Worksheet



16

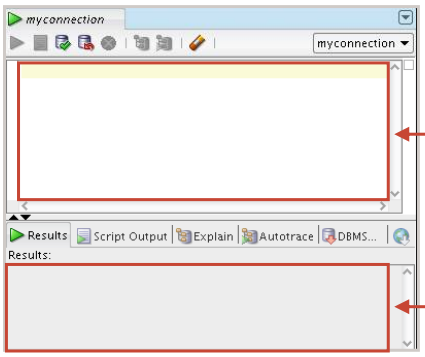
You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of the SQL statements that you have executed. You can use the SQL Worksheet toolbar that contains icons to perform the following tasks:

1. **Run Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all the statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
4. **Autotrace:** Generates trace information for the statement
5. **SQL Tuning Advisory:** Analyzes high-volume SQL statements and offers tuning recommendations
6. **Commit:** Writes any changes to the database and ends the transaction
7. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction
8. **Unshared SQL Worksheet:** Creates a separate unshared SQL Worksheet for a connection
9. **Clear:** Erases the statement or statements in the Enter SQL Statement box
10. **SQL History:** Displays a dialog box with information about the SQL statements that you have executed
11. **To Upper/Lower/InitCap:** Changes the selected text to uppercase, lowercase, or initcap, respectively

Oracle Internal & Oracle Academy Use Only

Using the SQL Worksheet

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



17

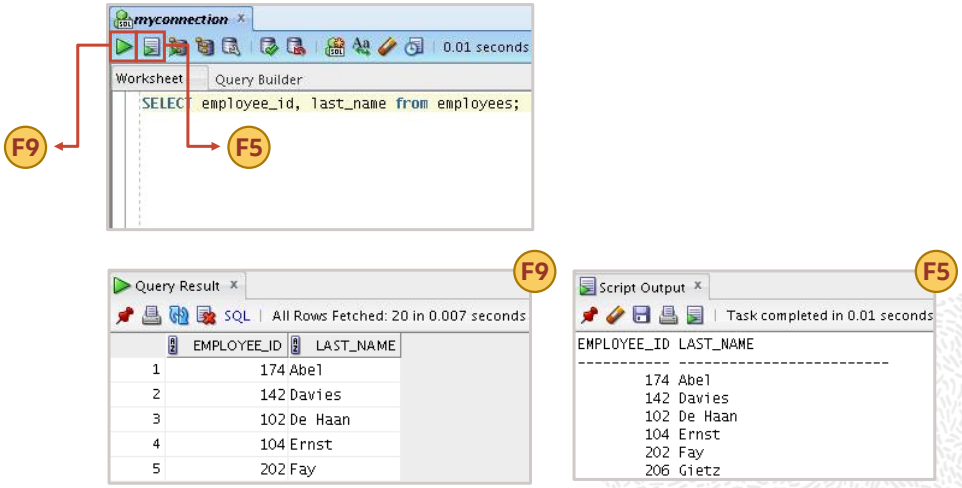
When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle database.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

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

Oracle Internal & Oracle Academy Use Only

Executing SQL Statements

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

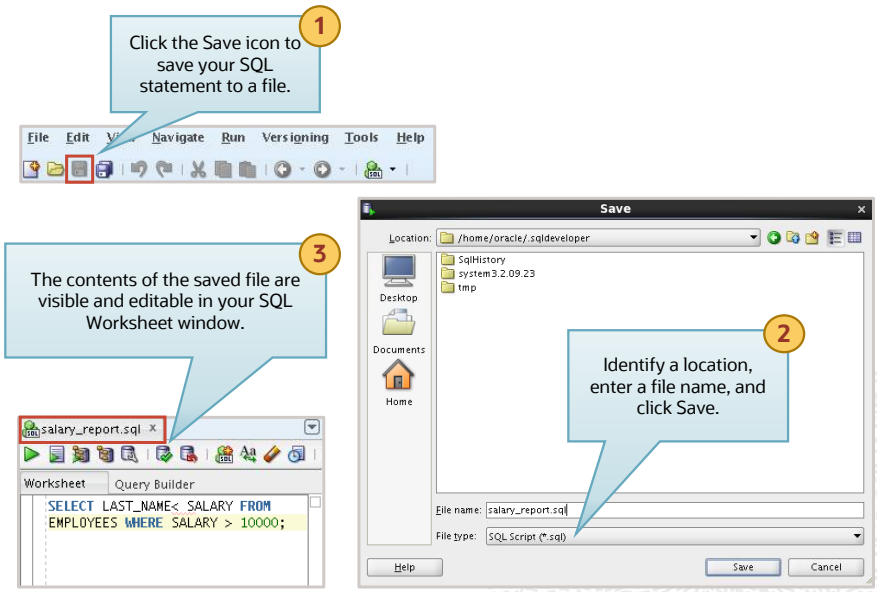


18

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

Oracle Internal & Oracle Academy Use Only

Saving SQL Scripts



19

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

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

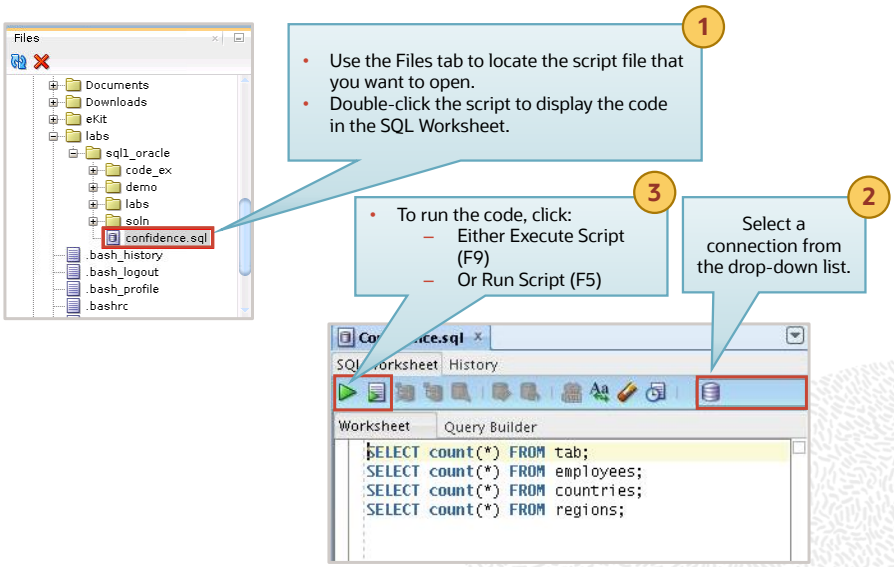
After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file is displayed as a tabbed page.

Script Pathing

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

Oracle Internal & Oracle Academy Use Only

Executing Saved Script Files: Method 1



20

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

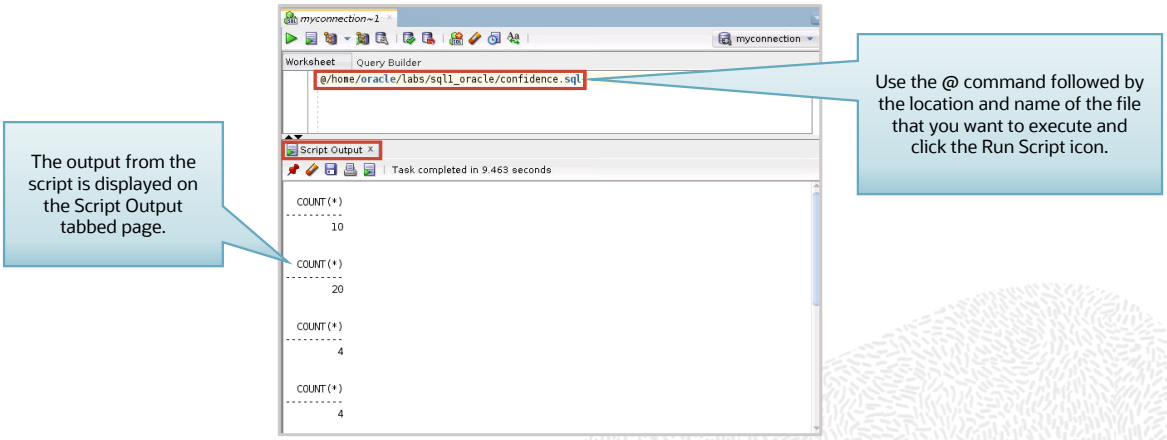
1. In the files navigator, select (or navigate to) the script file that you want to open.
2. Double-click the file to open it. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the connection drop-down list.
4. To run the code, click the Run Script (F5) icon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

Alternatively, you can also do the following:

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

Oracle Internal & Oracle Academy Use Only

Executing Saved Script Files: Method 2



21

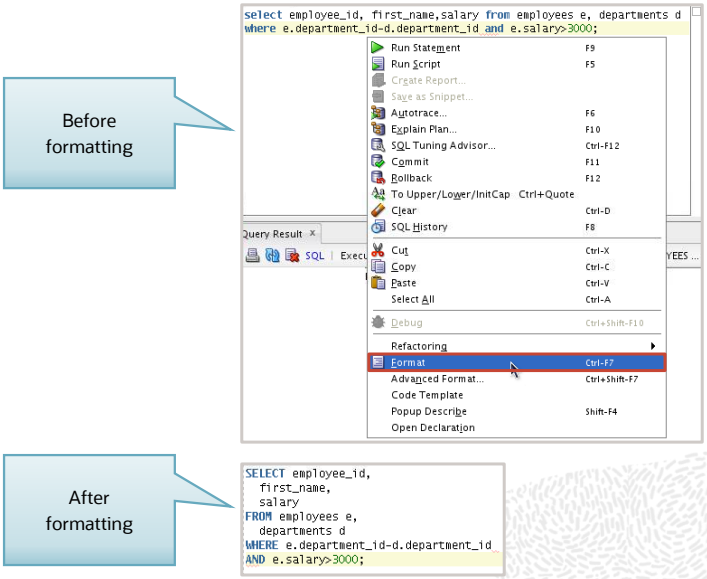
To run a saved SQL script, perform the following steps:

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

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

Oracle Internal & Oracle Academy Use Only

Formatting the SQL Code



22

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

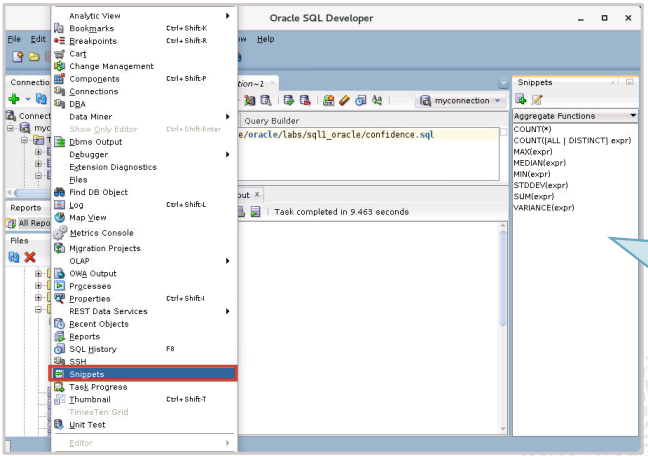
To format the SQL code, right-click in the statement area and select Format.

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

Oracle Internal & Oracle Academy Use Only

Using Snippets

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



The Snippets window is displayed on the right. From the drop-down list, you can select the functions category that you want.

23

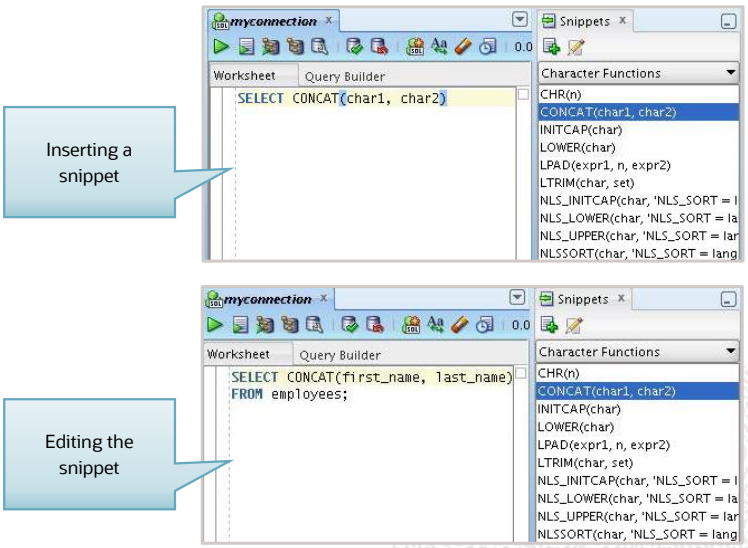
You may want to use certain code fragments when you use the SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has a feature called Snippets. Snippets are code fragments such as SQL functions, optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets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

To display Snippets, select View > Snippets.

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

Oracle Internal & Oracle Academy Use Only

Using Snippets: Example



24

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

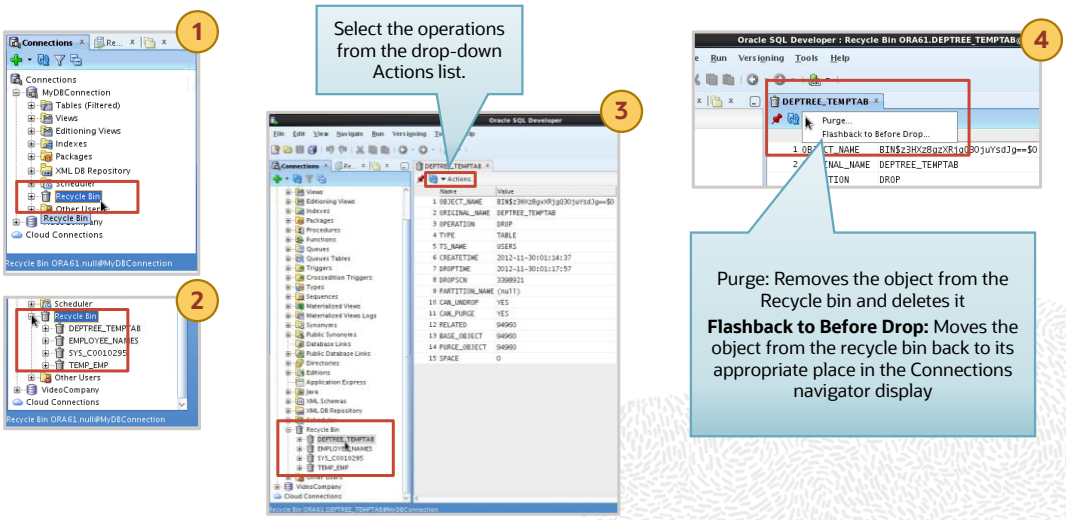
The example in the slide shows that `CONCAT(char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited, and the rest of the statement is added as in the following:

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

Oracle Internal & Oracle Academy Use Only

Using the Recycle Bin

The recycle bin holds objects that have been dropped.



25

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

To use the recycle bin, perform the following steps:

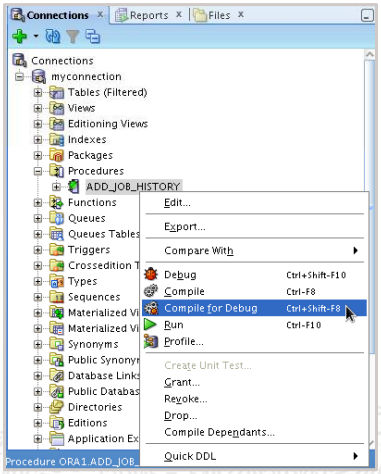
1. In the Connections navigator, select (or navigate to) the recycle bin.
2. Expand Recycle Bin and click the object name. The object details are displayed in the SQL Worksheet area.
3. Click the Actions drop-down list and select the operation that you want to perform on the object.

Oracle Internal & Oracle Academy Use Only

Debugging Procedures and Functions

Use:

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



26

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

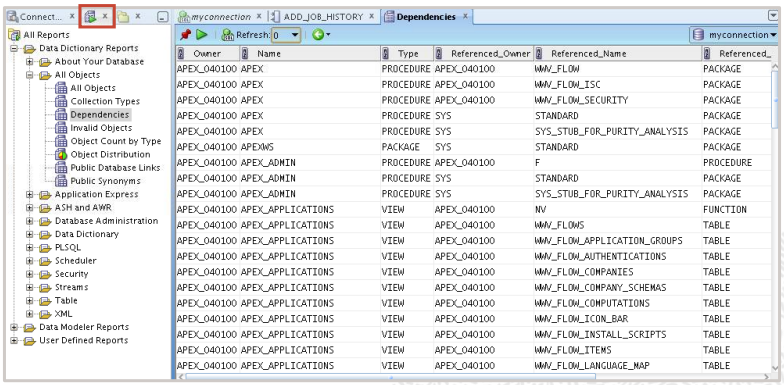
- **Find Execution Point** goes to the next execution point.
- **Resume** continues execution.
- **Step Over** bypasses the next method and goes to the next statement after the method.
- **Step Into** goes to the first statement in the next method.
- **Step Out** leaves the current method and goes to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

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

Oracle Internal & Oracle Academy Use Only

Database Reporting

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



27

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

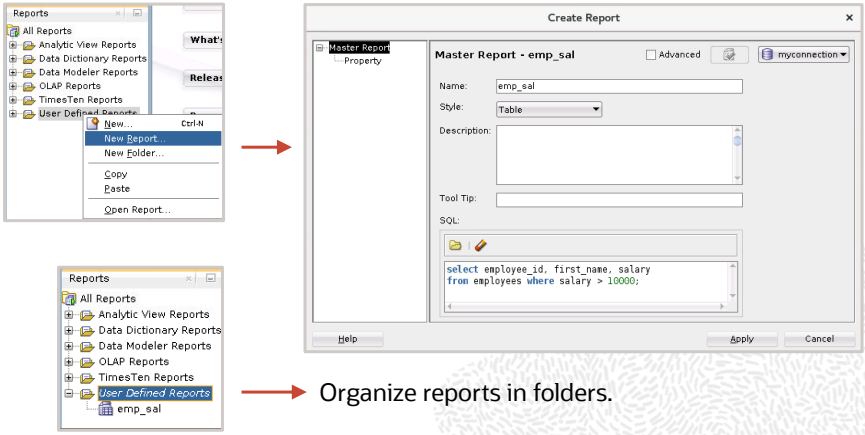
- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

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

Oracle Internal & Oracle Academy Use Only

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



Organize reports in folders.

28

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

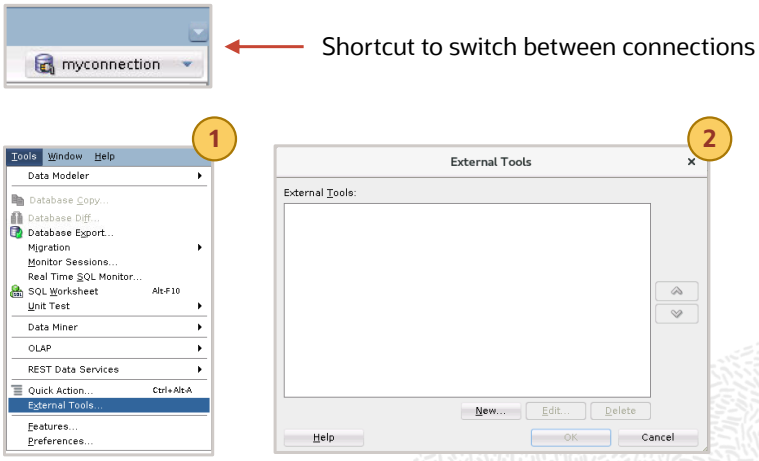
1. Right-click the User Defined Reports node under Reports and select Add Report.
2. In the Create Report dialog box, specify the report name and the SQL query to retrieve information for the report. Then click Apply.

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

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

Oracle Internal & Oracle Academy Use Only

External Tools



29

You have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver, available to you.

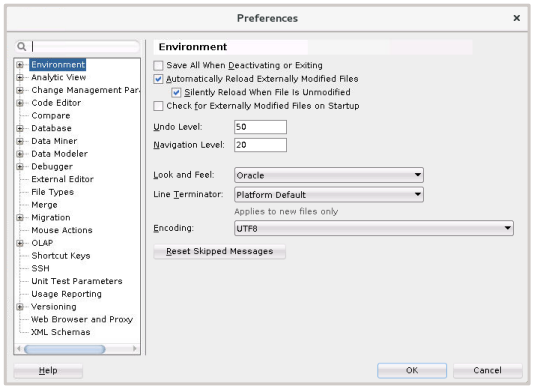
You can add external tools to the existing list or even delete shortcuts to the tools that you do not use frequently. To do so, perform the following steps:

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

Oracle Internal & Oracle Academy Use Only

Setting Preferences

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



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

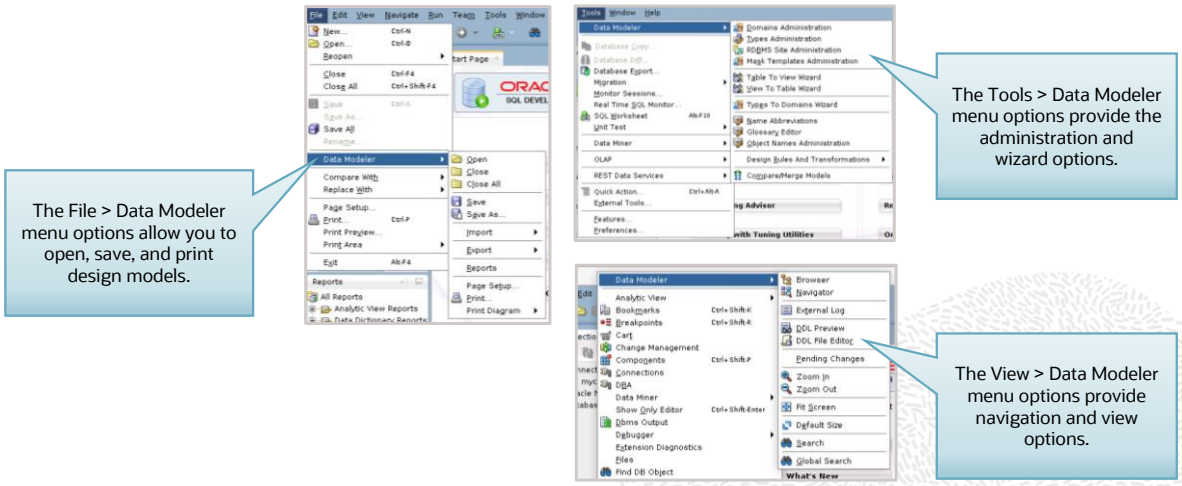
The preferences are grouped into the following categories:

- Environment
- Change Management Parameter
- Code Editor
- Compare and Merge
- Database
- Data Miner
- Data Modeler
- Debugger
- Extensions
- External Editor
- File Types
- Migration
- Mouseover Popups
- Shortcut Keys
- Unit Test Parameters
- Versioning
- Web Browser and Proxy
- XML Schemas

Oracle Internal & Oracle Academy Use Only

Data Modeler in SQL Developer

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



31

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

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

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

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to use SQL Developer to:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports
- Browse the data modeling options in SQL Developer

32

SQL Developer is a free graphical tool to simplify database development tasks. Using SQL Developer, you can browse, create, and edit database objects. You can use SQL Worksheet to run SQL statements and scripts. SQL Developer enables you to create and save your own special set of reports for repeated use.

Oracle Internal & Oracle Academy Use Only

ORACLE

C

Using SQL*Plus

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

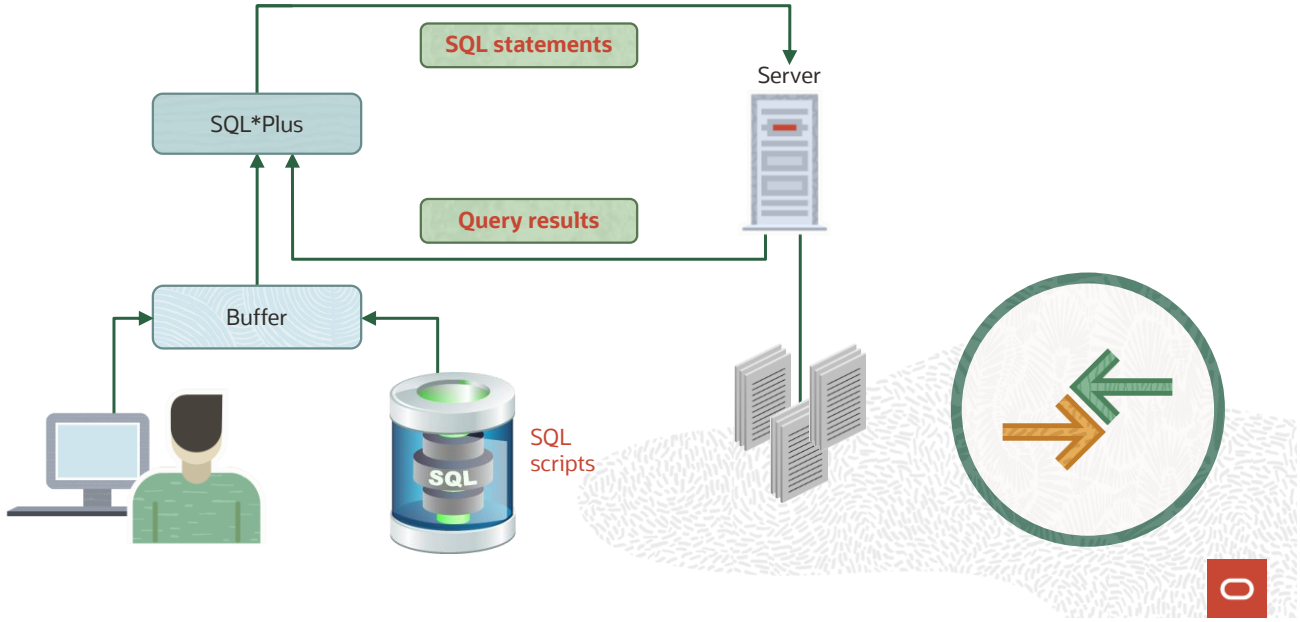
- Log in to SQL*Plus
- Edit SQL commands
- Format the output by using SQL*Plus commands
- Interact with script files

2

You might want to create `SELECT` statements that can be used repeatedly. This appendix covers the use of SQL*Plus commands to execute SQL statements. You learn how to format output using SQL*Plus commands, edit SQL commands, and save scripts in SQL*Plus.

Oracle Internal & Oracle Academy Use Only

SQL and SQL*Plus Interaction



3

SQL and SQL*Plus

SQL is a command language that is used for communication with the Oracle server from any tool or application. Oracle SQL contains many extensions. When you enter a SQL statement, it is stored in a part of the memory called the SQL buffer and remains there until you enter a new SQL statement. SQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle 19c Server for execution. It contains its own command language.

Features of SQL

- Can be used by a range of users, including those with little or no programming experience
- Is a nonprocedural language
- Reduces the amount of time required for creating and maintaining systems
- Is an English-like language

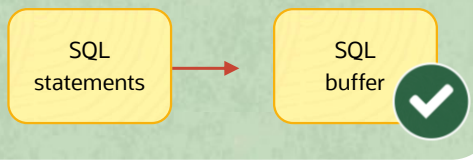
Features of SQL*Plus

- Accepts ad hoc entry of statements
- Accepts SQL input from files
- Provides a line editor for modifying SQL statements
- Controls environmental settings
- Formats query results into basic reports
- Accesses local and remote databases

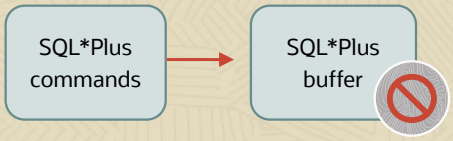
Oracle Internal & Oracle Academy Use Only

SQL Statements Versus SQL*Plus Commands

SQL
- A language
- ANSI-standard
- Keywords cannot be abbreviated.
- Statements manipulate data and table definitions in the database.



SQL*Plus
- An environment
- Oracle-proprietary
- Keywords can be abbreviated.
- Commands do not allow manipulation of values in the database.



4

The following table compares SQL and SQL*Plus:

SQL	SQL*Plus
Is a language for communicating with the Oracle server to access data	Recognizes SQL statements and sends them to the server
Is based on American National Standards Institute (ANSI)–standard SQL	Is the Oracle-proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Is entered into the SQL buffer on one or more lines	Is entered one line at a time, not stored in the SQL buffer
Does not have a continuation character	Uses a dash (–) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses a termination character to execute commands immediately	Does not require termination characters; executes commands immediately
Uses functions to perform some formatting	Uses commands to format data

Oracle Internal & Oracle Academy Use Only

SQL*Plus: Overview

- Log in to SQL*Plus.
- Describe the table structure.
- Edit your SQL statement.
- Execute SQL from SQL*Plus.
- Save SQL statements to files and append SQL statements to files.
- Execute saved files.
- Load commands from the file to buffer to edit.

5

SQL*Plus

SQL*Plus is an environment in which you can:

- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repeated use in the future

SQL*Plus commands can be divided into the following main categories:

Category	Purpose
Environment	Affect the general behavior of SQL statements for the session
Format	Format query results
File manipulation	Save, load, and run script files
Execution	Send SQL statements from the SQL buffer to the Oracle server
Edit	Modify SQL statements in the buffer
Interaction	Create and pass variables to SQL statements, print variable values, and print messages to the screen
Miscellaneous	Connect to the database, manipulate the SQL*Plus environment, and display column definitions

Oracle Internal & Oracle Academy Use Only

Logging In to SQL*Plus

```
sqlplus [username[/password[@database]]]
```

6

How you invoke SQL*Plus depends on the type of operating system that you are running Oracle Database on.

To log in from a Linux environment, perform the following steps:

1. Right-click your Linux desktop and select terminal.
2. Enter the `sqlplus` command shown in the slide.
3. Enter the username, password, and database name.

In the syntax:

`username`	Your database username
`password`	Your database password (Your password is visible if you enter it here.)
`@database`	The database connect string

Note: To ensure the integrity of your password, do not enter it at the operating system prompt. Instead, enter only your username. Enter your password at the password prompt.

Oracle Internal & Oracle Academy Use Only

Displaying the Table Structure

Use the SQL*Plus `DESCRIBE` command to display the structure of a table:

```
DESC[RIBE] tablename
```

7

In SQL*Plus, you can display the structure of a table by using the `DESCRIBE` command. The result of the command is a display of column names and data types, as well as an indication if a column must contain data.

In the syntax:

tablename	The name of any existing table, view, or synonym that is accessible to the user

To describe the `DEPARTMENTS` table, use the following command:

```
SQL> DESCRIBE DEPARTMENTS

Name                      Null     Type

----------------------- -------- ---------------


DEPARTMENT_ID             NOT NULL NUMBER(4)

DEPARTMENT_NAME           NOT NULL VARCHAR2(30)

MANAGER_ID                         NUMBER(6)

LOCATION_ID                        NUMBER(4)
```

Oracle Internal & Oracle Academy Use Only

Displaying the Table Structure

```
DESCRIBE departments
```

```
Name                      Null     Type
------------------------- -------- ------------
DEPARTMENT_ID             NOT NULL NUMBER(4)
DEPARTMENT_NAME           NOT NULL VARCHAR2(30)
MANAGER_ID                         NUMBER(6)
LOCATION_ID                        NUMBER(4)
```

8

The example in the slide displays information about the structure of the `DEPARTMENTS` table. In the result:

`Null`: Specifies whether a column must contain data (`NOT NULL` indicates that a column must contain data.)

`Type`: Displays the data type for a column

Oracle Internal & Oracle Academy Use Only

SQL*Plus Editing Commands

- A[PPEND] *text*
- C[HANGE] / *old* / *new*
- C[HANGE] / *text* /
- CL[EAR] BUFF[ER]
- DEL
- DEL *n*
- DEL *m n*

9

SQL*Plus commands are entered one line at a time and are not stored in the SQL buffer.

Command	Description
A[PPEND] *text*	Adds text to the end of the current line
C[HANGE] / *old* / *new*	Changes *old* text to *new* in the current line
C[HANGE] / *text* /	Deletes *text* from the current line
CL[EAR] BUFF[ER]	Deletes all lines from the SQL buffer
DEL	Deletes current line
DEL *n*	Deletes line *n*
DEL *m n*	Deletes lines *m* to *n* inclusive

Guidelines

- If you press Enter before completing a command, SQL*Plus prompts you with a line number.
- You terminate the SQL buffer either by entering one of the terminator characters (semicolon or slash) or by pressing Enter twice. The SQL prompt appears.

Oracle Internal & Oracle Academy Use Only

SQL*Plus Editing Commands

- `I[NPUT]`
- `I[NPUT] text`
- `L[IST]`
- `L[IST] n`
- `L[IST] m n`
- `R[UN]`
- `n`
- `n text`
- `0 text`

10

Command	Description
`I[NPUT]`	Inserts an indefinite number of lines
`I[NPUT] text`	Inserts a line consisting of *text*
`L[IST]`	Lists all lines in the SQL buffer
`L[IST] n`	Lists one line (specified by *n*)
`L[IST] m n`	Lists a range of lines (*m* to *n*) inclusive
`R[UN]`	Displays and runs the current SQL statement in the buffer
n	Specifies the line to make the current line
n text	Replaces line *n* with *text*
0 *text*	Inserts a line before line 1

Note: You can enter only one SQL*Plus command for each SQL prompt. SQL*Plus commands are not stored in the buffer. To continue a SQL*Plus command on the next line, end the first line with a hyphen (-).

Oracle Internal & Oracle Academy Use Only

Using LIST, n, and APPEND

```
LIST
  1  SELECT last_name
  2* FROM   employees
```

```
1
  1* SELECT last_name
```

```
A , job_id
  1* SELECT last_name, job_id
```

```
LIST
  1  SELECT last_name, job_id
  2* FROM   employees
```

11

- Use the L[IST] command to display the contents of the SQL buffer. The asterisk (*) beside line 2 in the buffer indicates that line 2 is the current line. Any edits that you made apply to the current line.
- Change the number of the current line by entering the number (n) of the line that you want to edit. The new current line is displayed.
- Use the A[PPEND] command to add text to the current line. The newly edited line is displayed. Verify the new contents of the buffer by using the LIST command.

Note: Many SQL*Plus commands, including LIST and APPEND, can be abbreviated to just their first letter. LIST can be abbreviated to L; APPEND can be abbreviated to A.

Oracle Internal & Oracle Academy Use Only

Using the CHANGE Command

```
LIST
  1* SELECT * from employees
```

```
c/employees/departments
  1* SELECT * from departments
```

```
LIST
  1* SELECT * from departments
```

12

- Use `L[IST]` to display the contents of the buffer.
- Use the `C[HANGE]` command to alter the contents of the current line in the SQL buffer. In this case, replace the `employees` table with the `departments` table. The new current line is displayed.
- Use the `L[IST]` command to verify the new contents of the buffer.

Oracle Internal & Oracle Academy Use Only

SQL*Plus File Commands

- SAVE *filename*
- GET *filename*
- START *filename*
- @ *filename*
- EDIT *filename*
- SPOOL *filename*
- EXIT

13

SQL statements communicate with the Oracle server. SQL*Plus commands control the environment, format query results, and manage files. You can use the commands described in the following table:

Command	Description
SAV[E] *filename* [.ext] [REP[LACE]APP[END]]	Saves the current contents of the SQL buffer to a file. Use APPEND to add to an existing file; use REPLACE to overwrite an existing file. The default extension is .sql.
GET *filename* [.ext]	Writes the contents of a previously saved file to the SQL buffer. The default extension for the file name is .sql.
STA[RT] *filename* [.ext]	Runs a previously saved command file
@ *filename*	Runs a previously saved command file (same as START)
ED[IT]	Invokes the editor and saves the buffer contents to a file named afiedt.buf
ED[IT] [*filename*[.ext]]	Invokes the editor to edit the contents of a saved file
SPO[OL] [*filename*[.ext]| OFF|OUT]	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the printer.
EXIT	Quits SQL*Plus

Oracle Internal & Oracle Academy Use Only

Using the SAVE and START Commands

```
LIST
  1  SELECT last_name, manager_id, department_id
  2* FROM employees
```

```
SAVE my_query
  Created file my_query
```

```
START my_query

LAST_NAME                 MANAGER_ID DEPARTMENT_ID
------------------------- ---------- -------------
King                                            90
Kochhar                          100            90
...
107 rows selected.
```

14

SAVE

Use the SAVE command to store the current contents of the buffer in a file. Thus, you can store frequently used scripts for use in the future.

START

Use the START command to run a script in SQL*Plus. You can also, alternatively, use the symbol @ to run a script.

```
@my_query
```

Oracle Internal & Oracle Academy Use Only

SERVEROUTPUT Command

- Use the SET SERVEROUT[PUT] command to control whether to display the output of stored procedures or PL/SQL blocks in SQL*Plus.
- The DBMS_OUTPUT line length limit is increased from 255 bytes to 32767 bytes.
- The default size is now unlimited.
- Resources are not pre-allocated when SERVEROUTPUT is set.
- Because there is no performance penalty, use UNLIMITED unless you want to conserve physical memory.

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNL[IMITED]}]
[FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NCATED]}]
```

15

Most of the PL/SQL programs perform input and output through SQL statements, to store data in database tables or query those tables. All other PL/SQL input/output is done through APIs that interact with other programs. For example, the DBMS_OUTPUT package has procedures, such as PUT_LINE. To see the result outside of PL/SQL requires another program, such as SQL*Plus, to read and display the data passed to DBMS_OUTPUT.

SQL*Plus does not display DBMS_OUTPUT data unless you first issue the SQL*Plus command SET SERVEROUTPUT ON as follows:

```
SET SERVEROUTPUT ON
```

Notes

- SIZE sets the number of bytes of the output that can be buffered within the Oracle Database server. The default is UNLIMITED. n cannot be less than 2000 or greater than 1,000,000.
- For additional information about SERVEROUTPUT, see *Oracle Database PL/SQL User's Guide and Reference 19c.*

Oracle Internal & Oracle Academy Use Only

Using the SQL*Plus SPOOL Command

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] | APP[END]] | OFF | OUT]
```

Option	Description
file_name[.ext]	Spools output to the specified file name
CRE[ATE]	Creates a new file with the name specified
REP[LACE]	Replaces the contents of an existing file. If the file does not exist, REPLACE creates the file.
APP[END]	Adds the contents of the buffer to the end of the file that you specify
OFF	Stops spooling
OUT	Stops spooling and sends the file to your computer's standard (default) printer

16

The SPOOL command stores query results in a file or optionally sends the file to a printer. The SPOOL command has been enhanced. You can now append to, or replace an existing file, where previously you could use SPOOL only to create (and replace) a file. REPLACE is the default.

To spool the output generated by commands in a script without displaying the output on screen, use SET TERMOUT OFF. SET TERMOUT OFF does not affect the output from commands that run interactively.

You must use quotation marks around file names that contain white space. To create a valid HTML file using SPOOL APPEND commands, you must use PROMPT or a similar command to create the HTML page header and footer. The SPOOL APPEND command does not parse HTML tags. SET SQLPLUSCOMPAT[IBILITY] to 9.2 or earlier to disable the CREATE, APPEND, and SAVE parameters.

Oracle Internal & Oracle Academy Use Only

Using the AUTOTRACE Command

- It displays a report after the successful execution of SQL DML statements such as SELECT, INSERT, UPDATE, or DELETE.
- The report can now include execution statistics and the query execution path.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
```

```
SET AUTOTRACE ON
-- The AUTOTRACE report includes both the optimizer
-- execution path and the SQL statement execution
-- statistics
```

17

EXPLAIN shows the query execution path by performing an EXPLAIN PLAN. STATISTICS displays SQL statement statistics. The formatting of your AUTOTRACE report may vary depending on the version of the server to which you are connected and the configuration of the server. The DBMS_XPLAN package provides an easy way to display the output of the EXPLAIN PLAN command in several predefined formats.

Notes

- For additional information about the package and subprograms, refer to *Oracle Database PL/SQL Packages and Types Reference 19c.*
- For additional information about the EXPLAIN PLAN, refer to *Oracle Database SQL Reference 19c*.
- For additional information about Execution Plans and the statistics, refer to *Oracle Database Performance Tuning Guide 19c*.

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to use SQL*Plus as an environment to do the following:

- Execute SQL statements
- Edit SQL statements
- Format the output
- Interact with script files

18

SQL*Plus is an execution environment that you can use to send SQL commands to the database server and to edit and save SQL commands. You can execute commands from the SQL prompt or from a script file.

Oracle Internal & Oracle Academy Use Only

ORACLE

D

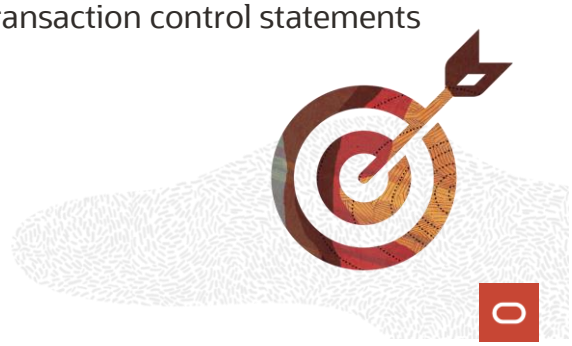
Commonly Used SQL Commands

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

- Execute a basic `SELECT` statement
- Create, alter, and drop a table using data definition language (DDL) statements
- Insert, update, and delete rows from one or more tables using data manipulation language (DML) statements
- Commit, roll back, and create savepoints by using transaction control statements
- Perform join operations on one or more tables



2

This lesson explains how to obtain data from one or more tables using the SELECT statement, to use DDL statements to alter the structure of data objects, to manipulate data in existing schema objects by using DML statements, to manage the changes made by DML statements, and to use joins to display data from multiple tables using SQL:1999 join syntax.

Note: Before the Oracle 9*i* release, the join syntax was proprietary. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax. Oracle Database 19c complies with ANSI/ISO SQL:2016 standards.

Oracle Internal & Oracle Academy Use Only

Basic SELECT Statement

- Use the SELECT statement to:
  - Identify the columns to be displayed
  - Retrieve data from one or more tables, object tables, views, object views, or materialized views
- A SELECT statement is also known as a query because it queries a database.
- Syntax:

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

3

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

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

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

Oracle Internal & Oracle Academy Use Only

SELECT Statement

- Select all columns:

```
SELECT *
FROM   job_history;
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA_REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

- Select specific columns:

```
SELECT manager_id, job_id
FROM   employees;
...
```

	MANAGER_ID	JOB_ID
1	(null)	AD_PRES
2	100	AD_VP
3	100	AD_VP
4	102	IT_PROG
5	103	IT_PROG
6	103	IT_PROG
7	100	ST_MAN
8	124	ST_CLERK
9	124	ST_CLERK
10	124	ST_CLERK

4

You can display all columns of data in a table by following the `SELECT` keyword with an asterisk (`*`) or by listing all the column names after the `SELECT` keyword. The first example in the slide displays all the rows from the `job_history` table. Specific columns of the table can be displayed by specifying the column names, separated by commas. The second example in the slide displays the `manager_id` and `job_id` columns from the `employees` table.

In the `SELECT` clause, specify the columns in the order in which you want them to appear in the output. For example, the following SQL statement displays the `location_id` column before displaying the `department_id` column:

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

Note: You can enter your SQL statement in a SQL Worksheet and click the Run Statement icon or press F9 to execute a statement in SQL Developer. The output displayed on the Results tabbed page appears as shown in the slide.

Oracle Internal & Oracle Academy Use Only

WHERE Clause

- Use the optional WHERE clause to:
  - Filter rows in a query
  - Produce a subset of rows
- Syntax:

```
SELECT * FROM    table
[WHERE   condition];
```

- Example:

```
SELECT location_id from departments
WHERE department_name = 'Marketing';
```

5

The WHERE clause specifies a condition to filter rows, producing a subset of the rows in the table. A condition specifies a combination of one or more expressions and logical (Boolean) operators. It returns a value of TRUE, FALSE, or NULL. The example in the slide retrieves the location_id of the marketing department.

The WHERE clause can also be used to update or delete data from the database.

Example:

```
UPDATE departments
SET  department_name = 'Administration'
WHERE department_id = 20;
and
DELETE from departments
WHERE department_id =20;
```

Oracle Internal & Oracle Academy Use Only

ORDER BY Clause

- Use the optional ORDER BY clause to specify the row order.
- Syntax:

```
SELECT * FROM     table
[WHERE    condition]
[ORDER BY {<column>|<position> } [ASC|DESC] [, ...] ];
```

- Example:

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

6

The ORDER BY clause specifies the order in which the rows should be displayed. The rows can be sorted in ascending or descending fashion. By default, the rows are displayed in ascending order.

The example in the slide retrieves rows from the employees table ordered first by ascending order of department_id and then by descending order of salary.

Oracle Internal & Oracle Academy Use Only

GROUP BY Clause

- Use the optional GROUP BY clause to group columns that have matching values into subsets.
- Each group has no two rows having the same value for the grouping column or columns.
- Syntax:

```
SELECT <column1, column2, ... column_n>
FROM  table
[WHERE    condition]
[GROUP BY <column> [, ...] ]
[ORDER BY <column> [, ...] ] ;
```

- Example:

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

7

The GROUP BY clause is used to group selected rows based on the value of expr(s) for each row. The clause groups rows but does not guarantee order of the result set. To order the groupings, use the ORDER BY clause.

Any SELECT list elements that are not included in aggregation functions must be included in the GROUP BY list of elements. This includes both columns and expressions. The database returns a single row of summary information for each group.

The example in the slide returns the minimum and maximum salaries for each department in the employees table.

Oracle Internal & Oracle Academy Use Only

Data Definition Language

- DDL statements are used to define, structurally change, and drop schema objects.
- The commonly used DDL statements are:
  - `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE`
  - `GRANT`, `REVOKE`
  - `TRUNCATE`

8

DDL statements enable you to alter the attributes of an object without altering the applications that access the object. You can also use DDL statements to alter the structure of objects while database users are performing work in the database. These statements are most frequently used to:

- Create, alter, and drop schema objects and other database structures, including the database itself and database users
- Delete all the data in schema objects without removing the structure of these objects
- Grant and revoke privileges and roles

Oracle Database implicitly commits the current transaction before and after every DDL statement.

Oracle Internal & Oracle Academy Use Only

CREATE TABLE Statement

- Use the CREATE TABLE statement to create a table in the database.
- Syntax:

```
CREATE TABLE tablename (
{column-definition | Table-level constraint}
[ , {column-definition | Table-level constraint} ] * )
```

- Example:

```
CREATE TABLE teach_dept (
department_id  NUMBER(3) PRIMARY KEY,
department_name VARCHAR2(10));
```

9

Use the CREATE TABLE statement to create a table in the database. To create a table, you must have the CREATE TABLE privilege and a storage area in which to create objects.

The table owner and the database owner automatically gain the following privileges on the table after it is created:

- INSERT
- SELECT
- REFERENCES
- ALTER
- UPDATE

The table owner and the database owner can grant the preceding privileges to other users.

Oracle Internal & Oracle Academy Use Only

ALTER TABLE Statement

- Use the ALTER TABLE statement to modify the definition of an existing table in the database.
- Example 1:

```
ALTER TABLE  teach_dept
ADD  location_id  NUMBER  NOT NULL;
```

- Example 2:

```
ALTER TABLE teach_dept
MODIFY department_name  VARCHAR2(30) NOT NULL;
```

10

The ALTER TABLE statement allows you to make changes to an existing table.

You can:

- Add a column to a table
- Add a constraint to a table
- Modify an existing column definition
- Drop a column from a table
- Drop an existing constraint from a table
- Increase the width of the VARCHAR and CHAR columns
- Change a table to have read-only status

Example 1 in the slide adds a new column called location_id to the teach_dept table.

Example 2 updates the existing department_name column from VARCHAR2(10) to VARCHAR2(30) and adds a NOT NULL constraint to it.

Oracle Internal & Oracle Academy Use Only

DROP TABLE Statement

- The DROP TABLE statement removes the table and all its data from the database.
- Example:

```
DROP TABLE teach_dept;
```

- DROP TABLE with the PURGE clause drops the table and releases the space that is associated with it.

```
DROP TABLE teach_dept PURGE;
```

- The CASCADE CONSTRAINTS clause drops all referential integrity constraints from the table.

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```

11

The DROP TABLE statement allows you to remove a table and its contents from the database and pushes it to the recycle bin. Dropping a table invalidates dependent objects and removes object privileges on the table.

Use the PURGE clause along with the DROP TABLE statement to release back to the tablespace the space allocated for the table. You cannot roll back a DROP TABLE statement with the PURGE clause, nor can you recover the table if you have dropped it with the PURGE clause.

The CASCADE CONSTRAINTS clause allows you to drop the reference to the primary key and unique keys in the dropped table.

Oracle Internal & Oracle Academy Use Only

GRANT Statement

- The GRANT statement assigns the privilege to:
  - Insert or delete data
  - Create a foreign key reference to the named table or to a subset of columns from a table
  - Select data, a view, or a subset of columns from a table
  - Create a trigger on a table
  - Execute a specified function or procedure
- Example:

```
GRANT SELECT any table to PUBLIC;
```

12

You can use the GRANT statement to:

- Assign privileges to a specific user or role, or to all users, to perform actions on database objects
- Grant a role to a user, to PUBLIC, or to another role

Before you issue a GRANT statement, check that the derby.database.sql authorization property is set to True. This property enables SQL Authorization mode. You can grant privileges on an object if you are the owner of the database.

You can grant privileges to all users by using the PUBLIC keyword. When PUBLIC is specified, the privileges or roles affect all current and future users.

Oracle Internal & Oracle Academy Use Only

Privilege Types

Assign the following privileges by using the `GRANT` statement:

- `ALL PRIVILEGES`
- `DELETE`
- `INSERT`
- `REFERENCES`
- `SELECT`
- `UPDATE`

13

Oracle Database provides a variety of privilege types to grant privileges to a user or role:

- Use the ALL PRIVILEGES privilege type to grant all privileges to the user or role for the specified table.
- Use the DELETE privilege type to grant permission to delete rows from the specified table.
- Use the INSERT privilege type to grant permission to insert rows into the specified table.
- Use the REFERENCES privilege type to grant permission to create a foreign key reference to the specified table.
- Use the SELECT privilege type to grant permission to perform SELECT statements on a table or view.
- Use the UPDATE privilege type to grant permission to use the UPDATE statement on the specified table.

Oracle Internal & Oracle Academy Use Only

REVOKE Statement

- Use the REVOKE statement to remove privileges from a user to perform actions on database objects.
- Revoke a *system privilege* from a user:

```
REVOKE DROP ANY TABLE
FROM hr;
```

- Revoke a *role* from a user:

```
REVOKE dw_manager
FROM sh;
```

14

The REVOKE statement removes privileges from a specific user (or users) or role to perform actions on database objects. It performs the following operations:

- Revokes a role from a user, from PUBLIC, or from another role
- Revokes privileges for an object if you are the owner of the object or the database owner

Note: To revoke a role or system privilege, you must have been granted the privilege with the ADMIN OPTION.

Oracle Internal & Oracle Academy Use Only

TRUNCATE TABLE Statement

- Use the TRUNCATE TABLE statement to remove all the rows from a table.
- Example:

```
TRUNCATE TABLE employees_demo;
```

- By default, Oracle Database performs the following tasks:
  - De-allocates space used by the removed rows
  - Sets the NEXT storage parameter to the size of the last extent removed from the segment by the truncation process

15

The TRUNCATE TABLE statement deletes all the rows from a specific table. Removing rows with the TRUNCATE TABLE statement can be more efficient than dropping and re-creating a table. Dropping and re-creating a table:

- Invalidates the dependent objects of the table
- Requires you to re-grant object privileges
- Requires you to re-create indexes, integrity constraints, and triggers
- Re-specify its storage parameters

The TRUNCATE TABLE statement spares you from these efforts.

Note: You cannot roll back a TRUNCATE TABLE statement.

Oracle Internal & Oracle Academy Use Only

Data Manipulation Language

- DML statements query or manipulate data in the existing schema objects.
- A DML statement is executed when:
  - New rows are added to a table by using the `INSERT` statement
  - Existing rows in a table are modified using the `UPDATE` statement
  - Existing rows are deleted from a table by using the `DELETE` statement
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

16

Data Manipulation Language (DML) statements enable you to query or change the contents of an existing schema object. These statements are most frequently used to:

- Add new rows of data to a table or view by specifying a list of column values or using a subquery to select and manipulate existing data
- Change column values in the existing rows of a table or view
- Remove rows from tables or views

A collection of DML statements that forms a logical unit of work is called a transaction. Unlike DDL statements, DML statements do not implicitly commit the current transaction.

Oracle Internal & Oracle Academy Use Only

INSERT Statement

- Use the INSERT statement to add new rows to a table.
- Syntax:

```
INSERT INTO    table [(column [, column...])]
VALUES       (value [, value...]);
```

- Example:

```
INSERT INTO    departments
VALUES      (200,'Development',104,1400);

1 rows inserted.
```

17

The INSERT statement adds rows to a table. Make sure to insert a new row containing values for each column and to list the values in the default order of the columns in the table. Optionally, you can also list the columns in the INSERT statement.

Example:

```
INSERT INTO job_history (employee_id, start_date,  end_date, job_id)
VALUES (120,'25-JUL-06','12-FEB_08','AC_ACCOUNT');
```

The syntax discussed in the slide allows you to insert a single row at a time. The VALUES keyword assigns the values of expressions to the corresponding columns in the column list.

Oracle Internal & Oracle Academy Use Only

UPDATE Statement Syntax

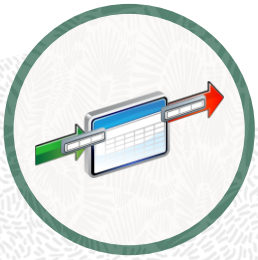
- Use the UPDATE statement to modify the existing rows in a table.
- Update more than one row at a time (if required).

```
UPDATE      table
SET    column = value [, column = value, ...]
[WHERE   condition];
```

- Example:

```
UPDATE  copy_emp
SET
22 rows updated
```

- Specify SET *column_name*= NULL to update a column value to NULL.



18

The UPDATE statement modifies the existing values in a table. Confirm the update operation by querying the table to display the updated rows. You can modify a specific row or rows by specifying the WHERE clause.

Example:

```
UPDATE employees
SET    salary = 17500
WHERE  employee_id = 102;
```

In general, use the primary key column in the WHERE clause to identify the row to update. For example, to update a specific row in the employees table, use employee_id to identify the row instead of employee_name, because more than one employee may have the same name.

Note: Typically, the condition keyword is composed of column names, expressions, constants, subqueries, and comparison operators.

Oracle Internal & Oracle Academy Use Only

DELETE Statement

- Use the DELETE statement to delete the existing rows from a table.
- Syntax:

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

- Write the DELETE statement using the WHERE clause to delete specific rows from a table.

```
DELETE FROM departments
WHERE  department_name = 'Finance';
1 rows deleted
```

19

The DELETE statement removes existing rows from a table. You must use the WHERE clause to delete a specific row or rows from a table based on the condition. The condition identifies the rows to be deleted. It may contain column names, expressions, constants, subqueries, and comparison operators.

The first example in the slide deletes the finance department from the departments table. You can confirm the delete operation by using the SELECT statement to query the table.

```
SELECT   *
FROM     departments
WHERE    department_name = 'Finance';
```

If you omit the WHERE clause, all rows in the table are deleted. Example:

```
DELETE FROM  copy_emp;
```

The preceding example deletes all the rows from the copy_emp table.

Oracle Internal & Oracle Academy Use Only

Transaction Control Statements

- Transaction control statements are used to manage the changes made by DML statements.
- The DML statements are grouped into transactions.
- Transaction control statements:
  - `COMMIT`
  - `ROLLBACK`
  - `SAVEPOINT`

20

A transaction is a sequence of SQL statements that Oracle Database treats as a single unit. Transaction control statements are used in a database to manage the changes made by DML statements and to group these statements into transactions.

Each transaction is assigned a unique `transaction_id`, and it groups SQL statements so that they are either all committed, which means they are applied to the database, or all rolled back, which means they are undone from the database.

Oracle Internal & Oracle Academy Use Only

COMMIT Statement

- Use the COMMIT statement to:
  - Permanently save the changes made to the database during the current transaction
  - Erase all savepoints in the transaction
  - Release transaction locks
- Example:

```
INSERT INTO    departments
VALUES     (201, 'Engineering', 106, 1400);
COMMIT;
```

```
1 rows inserted.
commited.
```

21

The COMMIT statement ends the current transaction by making all the pending data changes permanent. It releases all row and table locks and erases any savepoints that you may have marked since the last commit or rollback. The changes made using the COMMIT statement are visible to all users.

Oracle recommends that you explicitly end every transaction in your application programs with a COMMIT or ROLLBACK statement, including the last transaction, before disconnecting from Oracle Database. If you do not explicitly commit the transaction and the program terminates abnormally, the last uncommitted transaction is automatically rolled back.

Note: Oracle Database issues an implicit COMMIT before and after any data definition language (DDL) statement.

Oracle Internal & Oracle Academy Use Only

ROLLBACK Statement

- Use the ROLLBACK statement to undo changes made to the database during the current transaction.
- Use the TO SAVEPOINT clause to undo a part of the transaction after the savepoint.
- Example:

```
UPDATE          employees
SET             salary = 7000
WHERE           last_name = 'Ernst';
SAVEPOINT       Ernst_sal;

UPDATE          employees
SET             salary = 12000
WHERE           last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ersnt_sal;
```

22

The ROLLBACK statement undoes work done in the current transaction. To roll back the current transaction, no privileges are necessary.

Using ROLLBACK with the TO SAVEPOINT clause performs the following operations:

- Rolls back only the portion of the transaction after the savepoint
- Erases all savepoints created after that savepoint. The named savepoint is retained, so you can roll back to the same savepoint multiple times.

Using ROLLBACK without the TO SAVEPOINT clause performs the following operations:

- Ends the transaction
- Undoes all the changes in the current transaction
- Erases all savepoints in the transaction

Oracle Internal & Oracle Academy Use Only

SAVEPOINT Statement

- Use the SAVEPOINT statement to name and mark the current point in the processing of a transaction.
- Specify a name to each savepoint.
- Use distinct savepoint names within a transaction to avoid overriding.
- Syntax:

```
SAVEPOINT ;
```

23

The SAVEPOINT statement identifies a point in a transaction to which you can later roll back. You must specify a distinct name for each savepoint. If you create a second savepoint with the same identifier as an earlier savepoint, the earlier savepoint is erased.

After a savepoint has been created, you can continue processing, commit your work, roll back the entire transaction, or roll back to the savepoint.

A simple rollback or commit erases all savepoints. When you roll back to a savepoint, any savepoints marked after that savepoint are erased. The savepoint to which you have rolled back is retained.

When savepoint names are reused within a transaction, the Oracle Database moves (overrides) the savepoint from its old position to the current point in the transaction.

Oracle Internal & Oracle Academy Use Only

Joins

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

```
SELECT    table1.column, table2.column
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

- Write the join condition in the `WHERE` clause.
- Prefix the column name with the table name when the same column name appears in more than one table.

24

When data from more than one table in the database is required, a *join* condition is used. Rows in one table can be joined to rows in another table according to common values that exist in the corresponding columns (usually primary and foreign key columns).

To display data from two or more related tables, write a simple join condition in the `WHERE` clause.

In the syntax:

table1.column	Denotes the table and column from which data is retrieved
table1.column1 = table2.column2	Is the condition that joins (or relates) the tables together

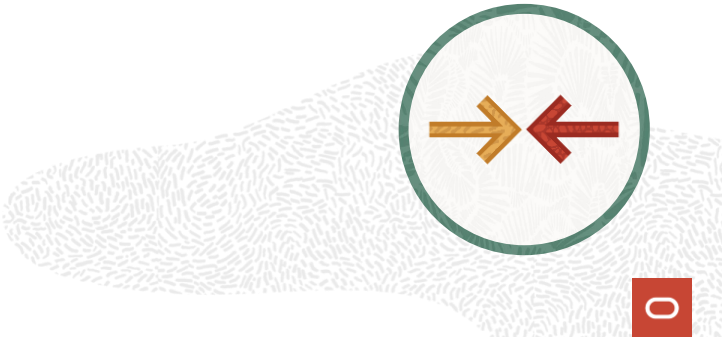
Guidelines

- When writing a `SELECT` statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join *n* tables together, you need a minimum of `n-1` join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

Oracle Internal & Oracle Academy Use Only

Types of Joins

- Natural join
- Equijoin
- Nonequijoin
- Outer join
- Self-join
- Cross join



25

To join tables, you can use Oracle's join syntax.

Note: Before the Oracle 9*i* release, the join syntax was proprietary. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax. Oracle Database 19c complies with ANSI/ISO SQL:2016 standards.

Oracle Internal & Oracle Academy Use Only

Qualifying Ambiguous Column Names

- Use table prefixes to:
  - Qualify column names that are in multiple tables
  - Improve performance
- Use table aliases instead of full table name prefixes.
- Table aliases give a table a shorter name.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

26

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the `DEPARTMENT_ID` column in the `SELECT` list could be from either the `DEPARTMENTS` table or the `EMPLOYEES` table. Therefore, it is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. However, using a table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. Therefore, you can use *table aliases*, instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, thereby using less memory.

The table name is specified in full, followed by a space, and then the table alias. For example, the `EMPLOYEES` table can be given an alias of e and the `DEPARTMENTS` table an alias of `d`.

Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the `FROM` clause, that table alias must be substituted for the table name throughout the `SELECT` statement.
- Table aliases should be meaningful.
- A table alias is valid only for the current `SELECT` statement.

Oracle Internal & Oracle Academy Use Only

Natural Join

- The NATURAL JOIN clause is based on all the columns in the two tables that have the same name.
- It selects rows from tables that have the same names and data values of columns.
- Example:

```
SELECT country_id, location_id, country_name,city
FROM countries NATURAL JOIN locations;
```

	COUNTRY_ID	LOCATION_ID	COUNTRY_NAME	CITY
1	US	1400	United States of America	Southlake
2	US	1500	United States of America	South San Francisco
3	US	1700	United States of America	Seattle
4	CA	1800	Canada	Toronto
5	UK	2500	United Kingdom	Oxford

27

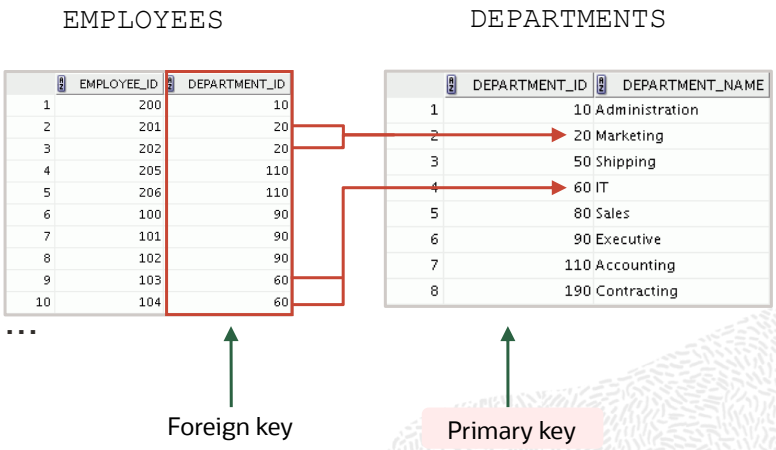
You can join tables automatically based on the columns in the two tables that have matching data types and names. You do this by using the NATURAL JOIN keywords.

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

In the example in the slide, the COUNTRIES table is joined to the LOCATIONS table by the COUNTRY_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.

Oracle Internal & Oracle Academy Use Only

Equijoins



28

An **equijoin** is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. To determine an employee's department name, you compare the values in the `DEPARTMENT_ID` column in the `EMPLOYEES` table with the `DEPARTMENT_ID` values in the `DEPARTMENTS` table. The relationship between the `EMPLOYEES` and `DEPARTMENTS` tables is an *equijoin*; that is, values in the `DEPARTMENT_ID` column in both tables must be equal. Often, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins*.

Oracle Internal & Oracle Academy Use Only

Retrieving Records with Equijoins

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	144	Vargas	50	50	1500
5	143	Matos	50	50	1500
6	142	Davies	50	50	1500
7	141	Rajs	50	50	1500
8	124	Mourgos	50	50	1500
9	103	Hunold	60	60	1400
10	104	Ernst	60	60	1400
11	107	Lorentz	60	60	1400

...

29

In the example in the slide:

- **The `SELECT` clause specifies the column names to retrieve:**
  - Employee last name, employee ID, and department ID, which are columns in the `EMPLOYEES` table
  - Department ID and location ID, which are columns in the `DEPARTMENTS` table
- **The `FROM` clause specifies the two tables that the database must access:**
  - `EMPLOYEES` table
  - `DEPARTMENTS` table
- **The `WHERE` clause specifies how the tables are to be joined:**

```
e.department_id = d.department_id
```

Because the `DEPARTMENT_ID` column is common to both tables, it must be prefixed with the table alias to avoid ambiguity. Other columns that are not present in both the tables need not be qualified by a table alias, but it is recommended for better performance.

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

Additional Search Conditions Using the AND No equijoin WHERE Operators

```
SELECT  d.department_id, d.department_name, l.city
FROM    departments d JOIN locations l
ON  d.location_id = l.location_id
AND d.department_id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

```
SELECT  d.department_id, d.department_name, l.city
FROM    departments d JOIN locations l
ON  d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```

30

In addition to the join, you may have criteria for your WHERE clause to restrict the rows in consideration for one or more tables in the join. The example in the slide performs a join on the DEPARTMENTS and LOCATIONS tables and, in addition, displays only those departments with ID equal to 20 or 50. 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 queries produce the same output.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

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	Fay	6000	C

...

31

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 grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits that are provided by the job grade table. That is, no employee earns less than the lowest value contained in the `LOWEST_SAL` column or more than the highest value contained in the `HIGHEST_SAL` column.

Note: Other conditions (such as <= and >=) can be used, but `BETWEEN` is the simplest. Remember to specify the low value first and the high value last when using 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

Retrieving Records by Using the USING Clause

- You can use the USING clause to match only one column when more than one column matches.
- You cannot specify this clause with a NATURAL join.
- Do not qualify the column name with a table name or table alias.
- Example:

```
SELECT country_id, country_name, location_id, city
FROM   countries JOIN locations
USING (country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	US	United States of America	1400	Southlake
2	US	United States of America	1500	South San Francisco
3	US	United States of America	1700	Seattle
4	CA	Canada	1800	Toronto
5	UK	United Kingdom	2500	Oxford

32

In the example in the slide, the COUNTRY_ID columns in the COUNTRIES and LOCATIONS tables are joined, and thus the LOCATION_ID of the location where an employee works is shown.

Oracle Internal & Oracle Academy Use Only

Retrieving Records by Using the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The ON clause makes code easy to understand.

```
SELECT e.employee_id, e.last_name, j.department_id,
FROM   employees e JOIN job_history j
ON     (e.employee_id = j.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	101	Kochhar	110
2	101	Kochhar	110
3	102	De Haan	60
4	176	Taylor	80
5	176	Taylor	80
6	200	Whalen	90
7	200	Whalen	90
8	201	Hartstein	20

33

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.

In this example, the EMPLOYEE_ID columns in the EMPLOYEES and JOB_HISTORY tables are joined using the ON clause. Wherever an employee ID in the EMPLOYEES table equals an employee ID in the JOB_HISTORY 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.employee_id = j.employee_id), are optional. So, even ON e.employee_id = j.employee_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 employee_ids.

Oracle Internal & Oracle Academy Use Only

Left Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the left table, is called a `LEFT OUTER JOIN`.
- Example:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM   countries c LEFT OUTER JOIN locations l
ON    (c.country_id = l.country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	CA	Canada	1800	Toronto
2	DE	Germany	(null)	(null)
3	UK	United Kingdom	2500	Oxford
4	US	United States of America	1400	Southlake
5	US	United States of America	1500	South San Francisco
6	US	United States of America	1700	Seattle

34

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

Oracle Internal & Oracle Academy Use Only

Right Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the right table, is called a `RIGHT OUTER JOIN`.
- Example:

```
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

...

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

35

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.

Oracle Internal & Oracle Academy Use Only

Full Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from both tables, is called a `FULL OUTER JOIN`.
- Example:

```
SELECT e.last_name, d.department_id, d.manager_id,
 d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON   (e.manager_id = d.manager_id) ;
```

	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	DEPARTMENT_NAME
1	King	(null)	(null)	(null)
2	Kochhar	90	100	Executive
3	De Haan	90	100	Executive
4	Hunold	(null)	(null)	(null)

...

19	Higgins	(null)	(null)	(null)
20	Gietz	110	205	Accounting
21	(null)	190	(null)	Contracting
22	(null)	10	200	Administration

36

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

Oracle Internal & Oracle Academy Use Only

Self-Join: Example

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker JOIN employees manager
ON worker.manager_id = manager.employee_id
ORDER BY worker.last_name;
```

	WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
1	Abel works for Zlotkey
2	Davies works for Mourgos
3	De Haan works for King
4	Ernst works for Hunold
5	Fay works for Hartstein
6	Gietz works for Higgins
7	Grant works for Zlotkey
8	Hartstein works for King
9	Higgins works for Kochhar

...

37

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. The example in the slide joins the `EMPLOYEES` table to itself. To simulate two tables in the `FROM` clause, there are two aliases, namely worker and manager, for the same table, `EMPLOYEES`.

In this example, the `WHERE` clause contains the join that means "where a worker's manager ID matches the employee ID for the manager."

Oracle Internal & Oracle Academy Use Only

Cross Join

- A `CROSS JOIN` is a `JOIN` operation that produces the Cartesian product of two tables.
- Example:

```
SELECT department_name, city
FROM department CROSS JOIN location;
```

	DEPARTMENT_NAME	CITY
1	Administration	Oxford
2	Administration	Seattle
3	Administration	South San Francisco
4	Administration	Southlake
5	Administration	Toronto
6	Marketing	Oxford
7	Marketing	Seattle
8	Marketing	South San Francisco
9	Marketing	Southlake
10	Marketing	Toronto

...

38

The `CROSS JOIN` syntax specifies the cross product. It is also known as a Cartesian product. A cross join produces the cross product of two relations and is essentially the same as the comma-delimited Oracle Database notation.

You do not specify any `WHERE` condition between the two tables in the `CROSS JOIN`.

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to use:

- The `SELECT` statement to retrieve rows from one or more tables
- DDL statements to alter the structure of objects
- DML statements to manipulate data in the existing schema objects
- Transaction control statements to manage the changes made by DML statements
- Joins to display data from multiple tables

39

There are many commonly used commands and statements in SQL. It includes the DDL statements, DML statements, transaction control statements, and joins.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

E

Generating Reports by Grouping Related Data

0

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to use:

- The `ROLLUP` operation to produce subtotal values
- The `CUBE` operation to produce cross-tabulation values
- The `GROUPING` function to identify the row values created by `ROLLUP` or `CUBE`
- `GROUPING SETS` to produce a single result set

2

In this appendix, you learn how to:

- Group data to obtain subtotal values by using the `ROLLUP` operator
- Group data to obtain cross-tabulation values by using the `CUBE` operator
- Use the `GROUPING` function to identify the level of aggregation in the result set produced by a `ROLLUP` or `CUBE` operator
- Use `GROUPING SETS` to produce a single result set that is equivalent to a `UNION ALL`

Oracle Internal & Oracle Academy Use Only

Review of Group Functions

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

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

- Example:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct),MAX(hire_date)
FROM   employees
WHERE  job_id LIKE 'SA%';
```

3

You can use the GROUP BY clause to divide the rows in a table into groups. You can then use group functions to return summary information for each group. Group functions can appear in select lists, and in ORDER BY and HAVING clauses. The Oracle server applies the group functions to each group of rows and returns a single result row for each group.

Types of group functions: Each of the group functions—AVG, SUM, MAX, MIN, COUNT, STDDEV, and VARIANCE—accepts one argument. The AVG, SUM, STDDEV, and VARIANCE functions operate only on numeric values. MAX and MIN can operate on numeric, character, or date data values. COUNT returns the number of non-NULL rows for the given expression. The example in the slide calculates the average salary, standard deviation on the salary, number of employees earning a commission, and the maximum hire date for those employees whose JOB_ID begins with SA.

Guidelines for Using Group Functions

- The data types for the arguments can be CHAR, VARCHAR2, NUMBER, or DATE.
- All group functions except COUNT(*) ignore null values. To substitute a value for null values, use the NVL function. COUNT returns either a number or zero.
- The Oracle server implicitly sorts the result set in ascending order of the grouping columns specified, when you use a GROUP BY clause. To override this default ordering, you can use DESC in an ORDER BY clause.

Oracle Internal & Oracle Academy Use Only

Review of the GROUP BY Clause

- Syntax:

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

- Example:

```
SELECT   department_id, job_id, SUM(salary),
         COUNT(employee_id)
FROM     employees
GROUP BY department_id, job_id ;
```

4

The example illustrated in the slide is evaluated by the Oracle server as follows:

- The SELECT clause specifies that the following columns be retrieved:
  - Department ID and job ID columns from the EMPLOYEES table
  - The sum of all the salaries and the number of employees in each group that you have specified in the GROUP BY clause
- The GROUP BY clause specifies how the rows should be grouped in the table. The total salary and the number of employees are calculated for each job ID within each department. The rows are grouped by department ID and then grouped by job within each department.

Oracle Internal & Oracle Academy Use Only

Review of the HAVING Clause

- Use the HAVING clause to specify which groups are to be displayed.
- You further restrict the groups on the basis of a limiting condition.

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

5

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

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

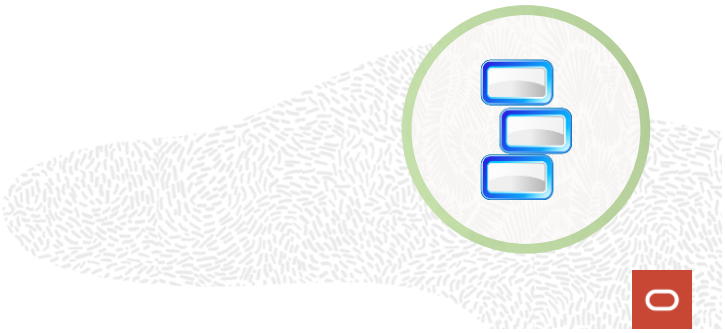
1. It groups rows.
2. It applies the group functions to the groups and displays the groups that match the criteria in the HAVING clause.

Oracle Internal & Oracle Academy Use Only

GROUP BY with ROLLUP and CUBE Operators

Use ROLLUP or CUBE with GROUP BY to produce superaggregate rows by cross-referencing columns.

- ROLLUP grouping produces a result set containing the regular grouped rows and the subtotal values.
- CUBE grouping produces a result set containing the rows from ROLLUP and cross-tabulation rows.



6

You specify ROLLUP and CUBE operators in the GROUP BY clause of a query. ROLLUP grouping produces a result set containing the regular grouped rows and subtotal rows. The ROLLUP operator also calculates a grand total. The CUBE operation in the GROUP BY clause groups the selected rows based on the values of all possible combinations of expressions in the specification and returns a single row of summary information for each group. You can use the CUBE operator to produce cross-tabulation rows.

Note: When working with ROLLUP and CUBE, make sure that the columns following the GROUP BY clause have meaningful, real-life relationships with each other; otherwise, the operators return irrelevant information.

Oracle Internal & Oracle Academy Use Only

ROLLUP Operator

- ROLLUP is an extension of the GROUP BY clause.
- Use the ROLLUP operation to produce cumulative aggregates, such as subtotals.

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

7

The ROLLUP operator delivers aggregates and superaggregates for expressions within a GROUP BY statement. The ROLLUP operator can be used by report writers to extract statistics and summary information from result sets. The cumulative aggregates can be used in reports, charts, and graphs.

The ROLLUP operator creates groupings by moving in one direction, from right to left, along the list of columns specified in the GROUP BY clause. It then applies the aggregate function to these groupings.

Note

- To produce subtotals in *n* dimensions (that is, *n* columns in the GROUP BY clause) without a ROLLUP operator, *n*+1 SELECT statements must be linked with UNION ALL. This makes the query execution inefficient because each of the SELECT statements causes table access. The ROLLUP operator gathers its results with just one table access. The ROLLUP operator is useful when there are many columns involved in producing the subtotals.
- Subtotals and totals are produced with ROLLUP. CUBE produces totals as well, but effectively rolls up in each possible direction, thereby producing cross-tabular data.

Oracle Internal & Oracle Academy Use Only

ROLLUP Operator: Example

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	10	(null)	4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20	(null)	19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30	(null)	24900
9	40	HR_REP	6500
10	40	(null)	6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50	(null)	156400
15	(null)	(null)	211200

1 2 3

8

In the example in the slide:

- Total salaries for every job ID within a department for those departments whose department ID is less than 60 are displayed by the GROUP BY clause
- The ROLLUP operator displays:
  - The total salary for each department whose department ID is less than 60
  - The total salary for all departments whose department ID is less than 60, irrespective of the job IDs

In this example, 1 indicates a group totaled by both DEPARTMENT_ID and JOB_ID, 2 indicates a group totaled only by DEPARTMENT_ID, and 3 indicates the grand total.

The ROLLUP operator creates subtotals that roll up from the most detailed level to a grand total, following the grouping list specified in the GROUP BY clause. First, it calculates the standard aggregate values for the groups specified in the GROUP BY clause (in the example, the sum of salaries grouped on each job within a department). Then it creates progressively higher-level subtotals, moving from right to left through the list of grouping columns. (In the example, the sum of salaries for each department is calculated, followed by the sum of salaries for all departments.)

- Given *n* expressions in the ROLLUP operator of the GROUP BY clause, the operation results in *n* + 1 (in this case, 2 + 1 = 3) groupings.
- Rows based on the values of the first *n* expressions are called rows or regular rows, and the others are called superaggregate rows.

Oracle Internal & Oracle Academy Use Only

CUBE Operator

- CUBE is an extension of the GROUP BY clause.
- Use the CUBE operator to produce cross-tabulation values with a single SELECT statement.

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

9

The CUBE operator is an additional switch in the GROUP BY clause in a SELECT statement. The CUBE operator can be applied to all aggregate functions, including AVG, SUM, MAX, MIN, and COUNT. It is used to produce result sets that are typically used for cross-tabular reports. ROLLUP produces only a fraction of possible subtotal combinations, whereas CUBE produces subtotals for all possible combinations of groupings specified in the GROUP BY clause, and a grand total.

The CUBE operator is used with an aggregate function to generate additional rows in a result set. Columns included in the GROUP BY clause are cross-referenced to produce a superset of groups. The aggregate function specified in the select list is applied to these groups to produce summary values for the additional superaggregate rows. The number of extra groups in the result set is determined by the number of columns included in the GROUP BY clause.

In fact, every possible combination of the columns or expressions in the GROUP BY clause is used to produce superaggregates. If you have *n* columns or expressions in the GROUP BY clause, there will be 2^n possible superaggregate combinations. Mathematically, these combinations form an *n*-dimensional cube, which is how the operator got its name.

By using application or programming tools, these superaggregate values can then be fed into charts and graphs that convey results and relationships visually and effectively.

Oracle Internal & Oracle Academy Use Only

CUBE Operator: Example

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)	
1	(null)	(null)	211200	1
2	(null)	HR_REP	6500	
3	(null)	MK_MAN	13000	
4	(null)	MK_REP	6000	
5	(null)	PU_MAN	11000	
6	(null)	ST_MAN	36400	2
7	(null)	AD_ASST	4400	
8	(null)	PU_CLERK	13900	
9	(null)	SH_CLERK	64300	
10	(null)	ST_CLERK	55700	
11	10	(null)	4400	
12	10	AD_ASST	4400	3
13	20	(null)	19000	
14	20	MK_MAN	13000	
15	20	MK_REP	6000	
16	30	(null)	24900	4

10

The output of the SELECT statement in the example can be interpreted as follows:

- The total salary for every job within a department (for departments with a department ID of lower than 60)
- The total salary for each department with a department ID of lower than 60
- The total salary for each job irrespective of the department
- The total salary for those departments with a department ID of lower than 60, irrespective of the job title

In this example, 1 indicates the grand total, 2 indicates the rows totaled by JOB_ID alone, 3 indicates some of the rows totaled by DEPARTMENT_ID and JOB_ID, and 4 indicates some of the rows totaled by DEPARTMENT_ID alone.

The CUBE operator has also performed the ROLLUP operation to display the subtotals for those departments whose department ID is less than 60 and the total salary for those departments whose department ID is less than 60, irrespective of the job titles. Further, the CUBE operator displays the total salary for every job irrespective of the department.

Note: Similar to the ROLLUP operator, producing subtotals in *n* dimensions (that is, *n* columns in the GROUP BY clause) without a CUBE operator requires that 2^n SELECT statements be linked with UNION ALL. Thus, a report with three dimensions requires $2^3 = 8$ SELECT statements to be linked with UNION ALL.

Oracle Internal & Oracle Academy Use Only

GROUPING Function

The GROUPING function:

- Is used with either the CUBE or ROLLUP operator
- Is used to find the groups forming the subtotal in a row
- Is used to differentiate stored NULL values from NULL values created by ROLLUP or CUBE
- Returns 0 or 1

```
SELECT    [column,] group_function(column) .. ,
          GROUPING(expr)
FROM       table
[WHERE     condition]
[GROUP BY [ROLLUP][CUBE] group_by_expression]
[HAVING    having_expression]
[ORDER BY column];
```

11

The GROUPING function can be used with either the CUBE or the ROLLUP operator to help you understand how a summary value has been obtained.

It uses a single column as its argument. The *expr* in the GROUPING function must match one of the expressions in the GROUP BY clause. The function returns a value of 0 or 1.

The values returned by the GROUPING function are useful to:

- Determine the level of aggregation of a given subtotal (that is, the group or groups on which the subtotal is based)
- Identify whether a NULL value in the expression column of a row of the result set indicates:
  - A NULL value from the base table (stored NULL value)
  - A NULL value created by ROLLUP or CUBE (as a result of a group function on that expression)

A value of 0 returned by the GROUPING function based on an expression indicates one of the following:

- The expression has been used to calculate the aggregate value.
- The NULL value in the expression column is a stored NULL value.

A value of 1 returned by the GROUPING function based on an expression indicates one of the following:

- The expression has not been used to calculate the aggregate value.
- The NULL value in the expression column is created by ROLLUP or CUBE as a result of grouping.

Oracle Internal & Oracle Academy Use Only

GROUPING Function: Example

```
SELECT   department_id DEPTID, job_id JOB,
         SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB
FROM     employees
WHERE    department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```

	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10	(null)	4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20	(null)	19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30	(null)	24900	0	1
9	40	HR_REP	6500	0	0
10	40	(null)	6500	0	1
11	(null)	(null)	54800	1	1

12

In the example in the slide, consider the summary value 4400 in the first row (labeled 1). This summary value is the total salary for the job ID of `AD_ASST` within department 10. To calculate this summary value, both the `DEPARTMENT_ID` and `JOB_ID` columns have been taken into account. Thus, a value of `0` is returned for both the `GROUPING(department_id)` and `GROUPING(job_id)` expressions.

Consider the summary value 4400 in the second row (labeled 2). This value is the total salary for department 10 and has been calculated by taking into account the `DEPARTMENT_ID` column; thus, a value of `0` has been returned by `GROUPING(department_id)`. Because the `JOB_ID` column has not been taken into account to calculate this value, a value of `1` has been returned for `GROUPING(job_id)`. You can observe similar output in the fifth row.

In the last row, consider the summary value 54800 (labeled 3). This is the total salary for those departments whose department ID is less than 50 and all job titles. To calculate this summary value, neither the `DEPARTMENT_ID` nor the `JOB_ID` columns has been taken into account. Thus, a value of `1` is returned for both `GROUPING(department_id)` and `GROUPING(job_id)` expressions.

Oracle Internal & Oracle Academy Use Only

GROUPING SETS

- The GROUPING SETS syntax is used to define multiple groupings in the same query.
  - All groupings specified in the GROUPING SETS clause are computed and the results of individual groupings are combined with a UNION ALL operation.
- Grouping set efficiency:
  - Only one pass over the base table is required.
  - There is no need to write complex UNION statements.
  - The more elements GROUPING SETS has, the greater is the performance benefit.

13

GROUPING SETS is a further extension of the GROUP BY clause that you can use to specify multiple groupings of data. Doing so facilitates efficient aggregation, which in turn facilitates analysis of data across multiple dimensions.

A single SELECT statement can now be written by using GROUPING SETS to specify various groupings (which can also include ROLLUP or CUBE operators), rather than multiple SELECT statements combined by UNION ALL operators.

For example:

```
SELECT department_id, job_id, manager_id, AVG(salary)
 FROM employees
 GROUP BY
 GROUPING SETS
 ((department_id, job_id, manager_id),
 (department_id, manager_id),(job_id, manager_id));
```

This statement calculates aggregates over three groupings:

(department_id, job_id, manager_id), (department_id, manager_id), and (job_id, manager_id)

Without this feature, multiple queries combined together with UNION ALL are required to obtain the output of the preceding SELECT statement. A multiquery approach is inefficient because it requires multiple scans of the same data.

Oracle Internal & Oracle Academy Use Only

Compare the previous example with the following alternative:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

This statement computes all the 8 (2 *2 *2) groupings, though only the `(department_id, job_id, manager_id)`, `(department_id, manager_id)`, and `(job_id, manager_id)` groups are of interest to you.

Another alternative is the following statement:

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

This statement requires three scans of the base table, which makes it inefficient.

`CUBE` and `ROLLUP` can be thought of as grouping sets with very specific semantics and results. The following equivalencies show this fact:

`CUBE(a, b, c)` is equivalent to	`GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())`
`ROLLUP(a, b,c)` is equivalent to	`GROUPING SETS ((a, b, c), (a, b),(a), ())`

Oracle Internal & Oracle Academy Use Only

GROUPING SETS: Example

```
SELECT   department_id, job_id,
         manager_id,AVG(salary)
FROM     employees
GROUP BY GROUPING SETS
((department_id,job_id), (job_id,manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
1	(null)	SH_CLERK	122	3200
2	(null)	AC_MGR	101	12000
3	(null)	ST_MAN	100	7280
4 ...	(null)	ST_CLERK	121	2675

1

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
39	110	AC_MGR	(null)	12000
40	90	AD_PRES	(null)	24000
41	60	IT_PROG	(null)	5760
42	100	FI_MGR	(null)	12000

...

2

15

The query in the slide calculates aggregates over two groupings. The table is divided into the following groups:

- Department ID, Job ID
- Job ID, Manager ID

The average salaries for each of these groups are calculated. The result set displays the average salary for each of the two groups.

In the output, the group marked as 1 can be interpreted as the following:

- The average salary of all employees with the `SH_CLERK` job ID under manager 122 is $ 3,200.
- The average salary of all employees with the `AC_MGR` job ID under manager 101 is $ 12,000, and so on.

The group marked as 2 in the output is interpreted as the following:

- The average salary of all employees with the `AC_MGR` job ID in department 110 is $ 12,000.
- The average salary of all employees with the `AD_PRES` job ID in department 90 is $ 24,000, and so on.

Oracle Internal & Oracle Academy Use Only

The example in the slide can also be written as:

```
SELECT department_id, job_id, NULL as manager_id,
       AVG(salary) as AVGSAL

FROM employees

GROUP BY department_id, job_id

UNION ALL

SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL

FROM employees

GROUP BY job_id, manager_id;
```

In the absence of an optimizer that looks across query blocks to generate the execution plan, the preceding query will need two scans of the base table `EMPLOYEES`. This could be very inefficient. Therefore, the usage of the `GROUPING SETS` statement is recommended.

Oracle Internal & Oracle Academy Use Only

Composite Columns

- A composite column is a collection of columns that are treated as a unit.

  `ROLLUP (a, (b, c), d)`

- Use parentheses within the `GROUP BY` clause to group columns so that they are treated as a unit while computing `ROLLUP` or `CUBE` operations.
- When used with `ROLLUP` or `CUBE`, composite columns would require skipping aggregation across certain levels.

17

A composite column is a collection of columns that are treated as a unit during the computation of groupings. You specify the columns in parentheses as in the following statement: `ROLLUP (a, (b, c), d)`

Here, `(b, c)` forms a composite column and is treated as a unit. In general, composite columns are useful in `ROLLUP`, `CUBE`, and `GROUPING SETS`. For example, in `CUBE` or `ROLLUP`, composite columns would require skipping aggregation across certain levels.

That is, `GROUP BY ROLLUP(a, (b, c) )` is equivalent to:

```
GROUP BY a, b, c UNION ALL

GROUP BY a UNION ALL

GROUP BY ()
```

Here, `(b, c)` is treated as a unit and `ROLLUP` is not applied across `(b, c)`. It is as though you have an alias—for example, `z` as an alias for `(b, c)`, and the `GROUP BY` expression reduces to: `GROUP BY ROLLUP(a, z)`.

Note: `GROUP BY( )` is typically a `SELECT` statement with `NULL` values for the columns `a` and `b` and only the aggregate function. It is generally used for generating grand totals.

```
SELECT   NULL, NULL, aggregate_col

FROM     <table_name>

GROUP BY ( );
```

Oracle Internal & Oracle Academy Use Only

Compare this with the normal `ROLLUP`.

`For example`:

```
GROUP BY ROLLUP(a, b, c)
```

This would be equivalent to:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

Similarly:

```
GROUP BY CUBE((a, b), c)
```

This would be equivalent to:

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP By ()
```

The following table shows the `GROUPING SETS` specification and the equivalent `GROUP BY` specification.

`GROUPING SETS` Statements	Equivalent `GROUP BY` Statements

`GROUP BY GROUPING SETS(a, b, c)`	`GROUP BY a UNION ALL`
	`GROUP BY b UNION ALL`
	`GROUP BY c`

`GROUP BY GROUPING SETS(a, b,(b, c))`
(The `GROUPING SETS` expression has a composite column.)
	`GROUP BY a UNION ALL`
	`GROUP BY b UNION ALL`
	`GROUP BY b, c`

`GROUP BY GROUPING SETS((a, b, c))`	`GROUP BY a, b, c`

`GROUP BY GROUPING SETS(a, (b), ())`	`GROUP BY a UNION ALL`
	`GROUP BY b UNION ALL`
	`GROUP BY ()`

`GROUP BY GROUPING SETS (a,ROLLUP(b, c))` (The GROUPING SETS expression has a composite column.)	`GROUP BY a UNION ALL` `GROUP BY ROLLUP(b, c)`

Oracle Internal & Oracle Academy Use Only

Composite Columns: Example

```
SELECT   department_id, job_id, manager_id,
         SUM(salary)
FROM     employees
GROUP BY ROLLUP( department_id,(job_id, manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	(null)	(null)	(null)	7000
3	10	AD_ASST	101	4400
4	10	(null)	(null)	4400
5	20	MK_MAN	100	13000
6	20	MK_REP	201	6000
7	20	(null)	(null)	19000

...

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
40	100	FI_MGR	101	12000
41	100	FI_ACCOUNT	108	39600
42	100	(null)	(null)	51600
43	110	AC_MGR	101	12000
44	110	AC_ACCOUNT	205	8300
45	110	(null)	(null)	20300
46	(null)	(null)	(null)	691400

19

Consider the example:

```
SELECT department_id, job_id,manager_id, SUM(salary)
 FROM   employees
 GROUP BY ROLLUP( department_id,job_id, manager_id);
```

This query results in the Oracle server computing the following groupings:

- `(job_id, manager_id)`
- `(department_id, job_id, manager_id)`
- `(department_id)`
- Grand total

If you are interested only in specific groups, you cannot limit the calculation to those groupings without using composite columns. With composite columns, this is possible by treating `JOB_ID` and `MANAGER_ID` columns as a single unit while rolling up. Columns enclosed in parentheses are treated as a unit while computing `ROLLUP` and `CUBE`. This is illustrated in the example in the slide. By enclosing the `JOB_ID` and `MANAGER_ID` columns in parentheses, you indicate to the Oracle server to treat `JOB_ID` and `MANAGER_ID` as a single unit, that is, as a composite column.

Oracle Internal & Oracle Academy Use Only

The example in the slide computes the following groupings:

- `department_id, job_id, manager_id)`
- `department_id)`
- `( )`

The example in the slide displays the following:

- Total salary for every job and manager (labeled 1)
- Total salary for every department, job, and manager (labeled 2)
- Total salary for every department (labeled 3)
- Grand total (labeled 4)

The example in the slide can also be written as:

```
SELECT       department_id, job_id, manager_id, SUM(salary)

FROM         employees

GROUP        BY department_id,job_id, manager_id

UNION        ALL

SELECT       department_id, TO_CHAR(NULL),TO_NUMBER(NULL),
     SUM(salary)

FROM         employees

GROUP BY     department_id

UNION ALL

SELECT  TO_NUMBER(NULL), TO_CHAR(NULL),TO_NUMBER(NULL), SUM(salary)

FROM    employees

GROUP BY ();
```

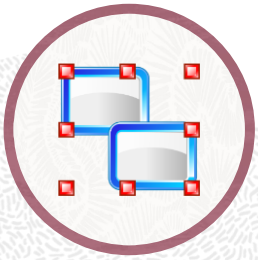
In the absence of an optimizer that looks across query blocks to generate the execution plan, the preceding query would need three scans of the base table, `EMPLOYEES`. This could be very inefficient. Therefore, the use of composite columns is recommended.

Oracle Internal & Oracle Academy Use Only

Concatenated Groupings

- Concatenated groupings offer a concise way to generate useful combinations of groupings.
- To specify concatenated grouping sets, you separate multiple grouping sets, `ROLLUP`, and `CUBE` operations with commas, so that the Oracle server combines them into a single `GROUP BY` clause.
- The result is a cross-product of groupings from each `GROUPING SET`.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```



21

Concatenated groupings offer a concise way to generate useful combinations of groupings. The concatenated groupings are specified by listing multiple grouping sets, `CUBE`s, and `ROLLUP`s and separating them with commas. The following is an example of concatenated grouping sets:

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

This SQL example defines the following groupings:

```
(a, c), (a, d), (b, c), (b, d)
```

Concatenation of grouping sets is very helpful for these reasons:

- **Ease of query development:** You need not manually enumerate all groupings.
- **Use by applications:** SQL generated by online analytical processing (OLAP) applications often involves concatenation of grouping sets, with each `GROUPING SET` defining groupings needed for a dimension.

Oracle Internal & Oracle Academy Use Only

Concatenated Groupings: Example

```
SELECT   department_id, job_id, manager_id,
         SUM(salary)
FROM     employees
GROUP BY department_id,
         ROLLUP(job_id),
         CUBE(manager_id);
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	10	AD_ASST	101	4400
3	20	MK_MAN	100	13000
4	20	MK_REP	201	6000
...				
1	90	AD_VP	100	34000
	90	AD_PRES	(null)	24000
...				
	(null)	SA_REP	(null)	7000
	10	AD_ASST	(null)	4400
...				
2	110	(null)	101	12000
	110	(null)	205	8300
	110	(null)	(null)	20300

3

22

The example in the slide results in the following groupings:

- `(department_id,job_id,)` (1)
- `(department_id,manager_id)` (2)
- `(department_id)` (3)

The total salary for each of these groups is calculated.

The following is another example of a concatenated grouping.

```
SELECT department_id, job_id, manager_id, SUM(salary) total
FROM employees
WHERE department_id<60
GROUP BY GROUPING SETS(department_id),
GROUPING SETS (job_id, manager_id);
```

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to use the:

- ROLLUP operation to produce subtotal values
- CUBE operation to produce cross-tabulation values
- GROUPING function to identify the row values created by ROLLUP or CUBE
- GROUPING SETS syntax to define multiple groupings in the same query
- GROUP BY clause to combine expressions in various ways:
  - Composite columns
  - Concatenated grouping sets

23

- ROLLUP and CUBE are extensions of the GROUP BY clause.
- ROLLUP is used to display subtotal and grand total values.
- CUBE is used to display cross-tabulation values.
- The GROUPING function enables you to determine whether a row is an aggregate produced by a CUBE or ROLLUP operator.
- With the GROUPING SETS syntax, you can define multiple groupings in the same query. GROUP BY computes all the groupings specified and combines them with UNION ALL.
- Within the GROUP BY clause, you can combine expressions in various ways:
  - To specify composite columns, you group columns within parentheses so that the Oracle server treats them as a unit while computing ROLLUP or CUBE operations.
  - To specify concatenated grouping sets, you separate multiple grouping sets, ROLLUP, and CUBE operations with commas so that the Oracle server combines them into a single GROUP BY clause. The result is a cross-product of groupings from each grouping set.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

F

Hierarchical Retrieval

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

- Interpret the concept of a hierarchical query
- Create a tree-structured report
- Format hierarchical data
- Exclude branches from the tree structure

2

In this appendix, you learn how to use hierarchical queries to create tree-structured reports.

Oracle Internal & Oracle Academy Use Only

Sample Data from the EMPLOYEES Table

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	100	King	AD_PRES	(null)
2	101	Kochhar	AD_VP	100
3	102	De Haan	AD_VP	100
4	103	Hunold	IT_PROG	102
5	104	Ernst	IT_PROG	103
6	107	Lorentz	IT_PROG	103

...

16	200	Whalen	AD_ASST	101
17	201	Hartstein	MK_MAN	100
18	202	Fay	MK_REP	201
19	205	Higgins	AC_MGR	101
20	206	Gietz	AC_ACCOUNT	205

3

Using hierarchical queries, you can retrieve data based on a natural hierarchical relationship between the rows in a table. A relational database does not store records in a hierarchical way. However, where a hierarchical relationship exists between the rows of a single table, a process called *tree walking* enables the hierarchy to be constructed. A hierarchical query is a method of reporting, with the branches of a tree in a specific order.

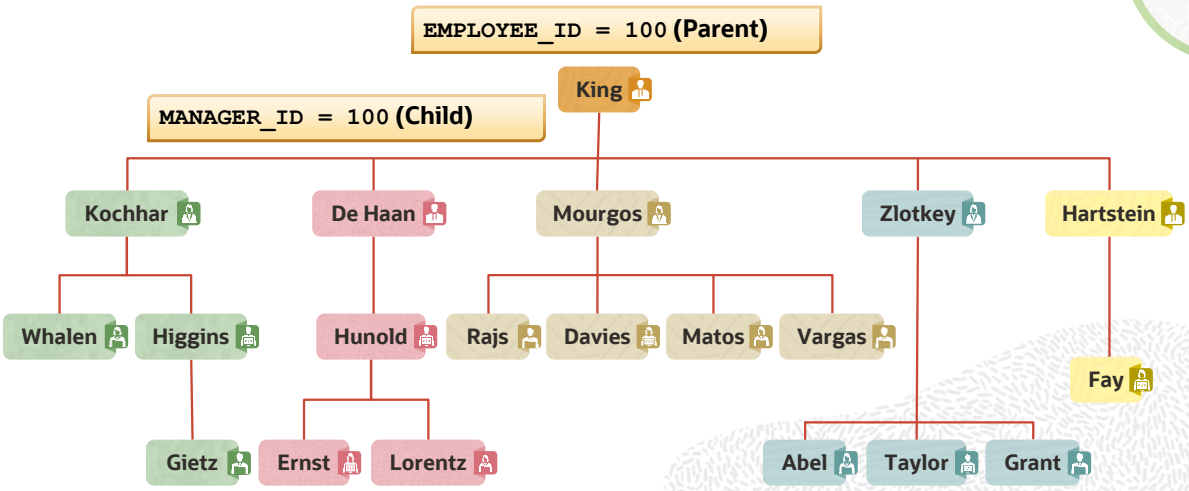
Imagine a family tree with the eldest members of the family found close to the base or trunk of the tree and the youngest members representing branches of the tree. Branches can have their own branches, and so on.

A hierarchical query is possible when a relationship exists between rows in a table. For example, in the slide, you see that Kochhar, De Haan, and Hartstein report to `MANAGER_ID` 100, which is King's `EMPLOYEE_ID`.

Note: Hierarchical trees are used in various fields such as human genealogy (family trees), livestock (breeding purposes), corporate management (management hierarchies), manufacturing (product assembly), evolutionary research (species development), and scientific research.

Oracle Internal & Oracle Academy Use Only

Natural Tree Structure



4

The EMPLOYEES table has a tree structure representing the management reporting line. The hierarchy can be created by looking at the relationship between equivalent values in the EMPLOYEE_ID and MANAGER_ID columns. This relationship can be exploited by joining the table to itself. The MANAGER_ID column contains the employee number of the employee's manager.

The parent/child relationship of a tree structure enables you to control:

- The direction in which the hierarchy is walked
- The starting point inside the hierarchy

Note: The slide displays an inverted tree structure of the management hierarchy of the employees in the EMPLOYEES table.

Oracle Internal & Oracle Academy Use Only

Hierarchical Queries

```
SELECT [LEVEL], column, expr...
FROM   table
[WHERE condition(s)]
[START WITH condition(s)]
[CONNECT BY PRIOR condition(s)] ;
```

condition:

```
expr comparison_operator expr
```

5

Keywords and Clauses

Hierarchical queries can be identified by the presence of the `CONNECT BY` and `START WITH` clauses.

In the syntax:

`SELECT`	Is the standard `SELECT` clause
`LEVEL`	For each row returned by a hierarchical query, the `LEVEL` pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on.
`FROM` *table*	Specifies the table, view, or snapshot containing the columns. You can select from only one table.
`WHERE`	Restricts the rows returned by the query without affecting other rows of the hierarchy
condition	Is a comparison with expressions
`START WITH`	Specifies the root rows of the hierarchy (where to start). This clause is required for a true hierarchical query.
`CONNECT BY PRIOR`	Specifies the columns in which the relationship between parent and child `PRIOR` rows exist. This clause is required for a hierarchical query.

Oracle Internal & Oracle Academy Use Only

Walking the Tree

Starting Point

- Specifies the condition that must be met
- Accepts any valid condition

```
START WITH column1 = value
```

- Using the EMPLOYEES table, start with the employee whose last name is Kochhar.

```
...START WITH last_name = 'Kochhar'
```

6

The row or rows to be used as the root of the tree are determined by the `START WITH` clause. The `START WITH` clause can contain any valid condition.

Examples

Using the `EMPLOYEES` table, start with King, the president of the company.

```
... START WITH manager_id IS NULL
```

Using the `EMPLOYEES` table, start with employee Kochhar. A `START WITH` condition can contain a subquery.

```
... START WITH employee_id = (SELECT employee_id
                                     FROM   employees
                                     WHERE  last_name =A 'Kochhar')
```

If the `START WITH` clause is omitted, the tree walk is started with all the rows in the table as root rows.

Note: The `CONNECT BY` and `START WITH` clauses are not American National Standards Institute (ANSI) SQL standard.

Oracle Internal & Oracle Academy Use Only

Walking the Tree

```
CONNECT BY PRIOR column1 = column2
```

Walk from the top down, using the EMPLOYEES table.

```
... CONNECT BY PRIOR employee_id = manager_id
```

Direction

Top down → Column1 = Parent Key
Column2 = Child Key

Bottom up → Column1 = Child Key
Column2 = Parent Key

7

The direction of the query is determined by the CONNECT BY PRIOR column placement. For top-down, the PRIOR operator refers to the parent row. For bottom-up, the PRIOR operator refers to the child row. To find the child rows of a parent row, the Oracle server evaluates the PRIOR expression for the parent row and the other expressions for each row in the table. Rows for which the condition is true are the child rows of the parent. The Oracle server always selects child rows by evaluating the CONNECT BY condition with respect to a current parent row.

Examples

Walk from the top down using the EMPLOYEES table. Define a hierarchical relationship in which the EMPLOYEE_ID value of the parent row is equal to the MANAGER_ID value of the child row:

```
... CONNECT BY PRIOR employee_id = manager_id
```

Walk from the bottom up using the EMPLOYEES table:

```
... CONNECT BY PRIOR manager_id = employee_id
```

The PRIOR operator does not necessarily need to be coded immediately following CONNECT BY. Thus, the following CONNECT BY PRIOR clause gives the same result as the one in the preceding example:

```
... CONNECT BY employee_id = PRIOR manager_id
```

Note: The CONNECT BY clause cannot contain a subquery.

Oracle Internal & Oracle Academy Use Only

Walking the Tree: From the Bottom Up

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START  WITH  employee_id = 101
CONNECT BY PRIOR manager_id = employee_id ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	101	Kochhar	AD_VP	100
2	100	King	AD_PRES	(null)

8

The example in the slide displays a list of managers starting with the employee whose employee ID is 101.

Oracle Internal & Oracle Academy Use Only

Walking the Tree: From the Top Down

```
SELECT  last_name||' reports to '||
PRIOR   last_name "Walk Top Down"
FROM    employees
START   WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id ;
```

	Walk Top Down
1	King reports to
2	King reports to
3	Kochhar reports to King
4	Greenberg reports to Kochhar
5	Faviet reports to Greenberg

...

105	Grant reports to Zlotkey
106	Johnson reports to Zlotkey
107	Hartstein reports to King
108	Fay reports to Hartstein

9

Walking from the top down, display the names of the employees and their manager. Use employee King as the starting point. Print only one column.

Example

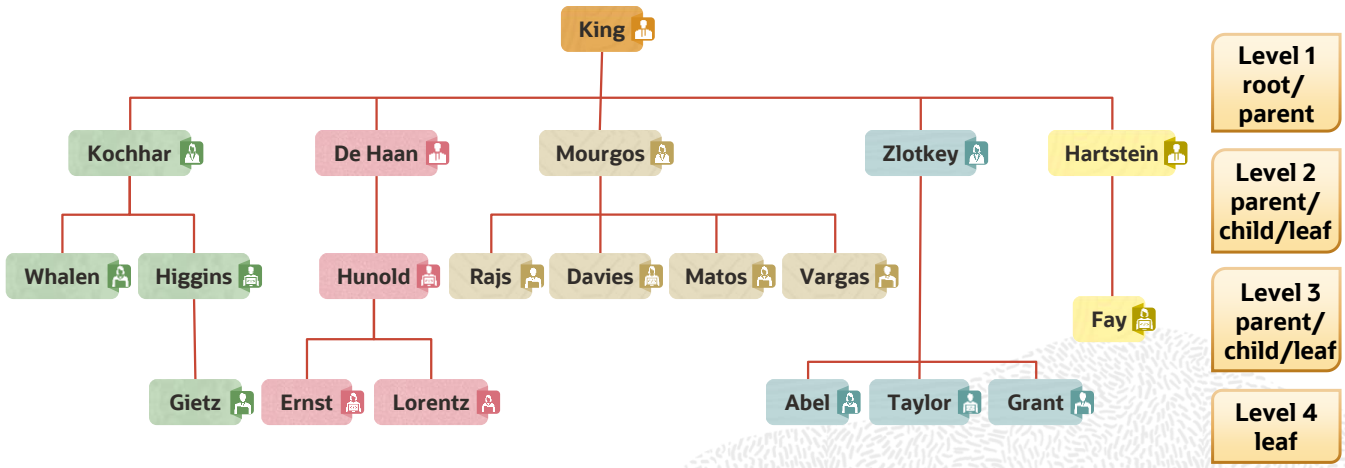
In the following example, `EMPLOYEE_ID` values are evaluated for the parent row and `MANAGER_ID` and `SALARY` values are evaluated for the child rows. The `PRIOR` operator applies only to the `EMPLOYEE_ID` value.

```
... CONNECT BY PRIOR employee_id = manager_id
                          AND salary > 15000;
```

To qualify as a child row, a row must have a `MANAGER_ID` value equal to the `EMPLOYEE_ID` value of the parent row and must have a `SALARY` value greater than $15,000.

Oracle Internal & Oracle Academy Use Only

Ranking Rows with the LEVEL Pseudocolumn



10

You can explicitly show the rank or level of a row in the hierarchy by using the LEVEL pseudocolumn. This will make your report more readable. The forks where one or more branches split away from a larger branch are called nodes, and the very end of a branch is called a leaf or leaf node. The graphic in the slide shows the nodes of the inverted tree with their LEVEL values. For example, employee Higgens is a parent and a child, whereas employee Davies is a child and a leaf.

LEVEL Pseudocolumn

Value	Level for Top Down	Level for Bottom up
1	A root node	A root node
2	A child of a root node	The parent of a root node
3	A child of a child, and so on	A parent of a parent, and so on

In the slide, King is the root or parent (LEVEL = 1). Kochhar, De Haan, Mourgos, Zlotkey, Hartstein, Higgens, and Hunold are children and also parents (LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant, and Fay are children and leaves (LEVEL = 3 and LEVEL = 4).

Note: A *root node* is the highest node within an inverted tree. A *child node* is any nonroot node. A parent node is any node that has children. A leaf node is any node without children. The number of levels returned by a hierarchical query may be limited by available user memory.

Oracle Internal & Oracle Academy Use Only

Formatting Hierarchical Reports Using LEVEL and LPAD

Create a report displaying company management levels, beginning with the highest level and indenting each of the following levels.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')
       AS org_chart
FROM   employees
START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

11

The nodes in a tree are assigned level numbers from the root. Use the LPAD function in conjunction with the LEVEL pseudocolumn to display a hierarchical report as an indented tree.

In the example in the slide:

- LPAD(*char1*,*n* [,*char2*]) returns *char1*, left-padded to length *n* with the sequence of characters in *char2*. The argument *n* is the total length of the return value as it is displayed on your terminal screen.
- LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_') defines the display format
- *char1* is the LAST_NAME, *n* the total length of the return value, is length of the LAST_NAME +(LEVEL*2)-2, and *char2* is '_'

That is, this tells SQL to take the LAST_NAME and left-pad it with the '_' character until the length of the resultant string is equal to the value determined by LENGTH(last_name)+(LEVEL*2)-2.

For King, LEVEL = 1. Therefore, (2 * 1) – 2 = 2 – 2 = 0. So King does not get padded with any '_' character and is displayed in column 1.

Oracle Internal & Oracle Academy Use Only

For Kochhar, `LEVEL = 2`. Therefore, (2 * 2) – 2 = 4 – 2 = 2. So Kochhar gets padded with 2 `'_'` characters and is displayed indented.
The rest of the records in the `EMPLOYEES` table are displayed similarly.

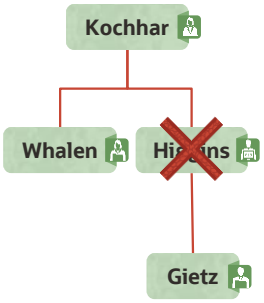
	ORG_CHART
1	King
2	__Kochhar
3	____Greenberg
4	______Faviet
5	______Chen
6	______Sciarra
7	______Urman
8	______Popp
9	____Whalen
10	____Mavris
11	____Baer
12	____Higgins
13	______Gietz
14	__De Haan
15	____Hunold
16	______Ernst
17	______Austin

Oracle Internal & Oracle Academy Use Only

Pruning Branches

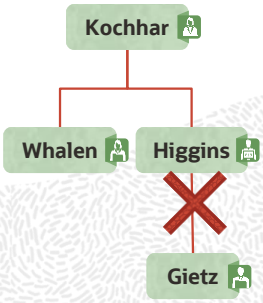
Use the WHERE clause to eliminate a node.

```
WHERE last_name != 'Higgins'
```



Use the CONNECT BY clause to eliminate a branch.

```
CONNECT BY PRIOR
employee_id = manager_id
AND last_name != 'Higgins'
```



13

You can use the WHERE and CONNECT BY clauses to prune the tree (that is, to control which nodes or rows are displayed). The predicate you use acts as a Boolean condition.

Examples

Starting at the root, walk from the top down and eliminate employee Higgins in the result, but process the child rows.

```
SELECT  department_id, employee_id,last_name, job_id, salary
FROM    employees
WHERE   last_name  != 'Higgins'
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

Starting at the root, walk from the top down and eliminate employee Higgins and all child rows.

```
SELECT  department_id, employee_id,last_name, job_id, salary
FROM    employees
START   WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND     last_name != 'Higgins';
```

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to:

- Use hierarchical queries to view a hierarchical relationship between rows in a table
- Specify the direction and starting point of the query
- Eliminate nodes or branches by pruning

14

You can use hierarchical queries to retrieve data based on a natural hierarchical relationship between rows in a table. The `LEVEL` pseudocolumn counts how far down a hierarchical tree you have traveled. You can specify the direction of the query using the `CONNECT BY PRIOR` clause. You can specify the starting point using the `START WITH` clause. You can use the `WHERE` and `CONNECT BY` clauses to prune the tree branches.

Oracle Internal & Oracle Academy Use Only

ORACLE

G

Writing Advanced Scripts

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

- Describe the type of problems that are solved by using SQL to generate SQL
- Create a basic SQL script
- Capture the output in a file
- Dump the contents of a table to a file
- Generate a dynamic predicate

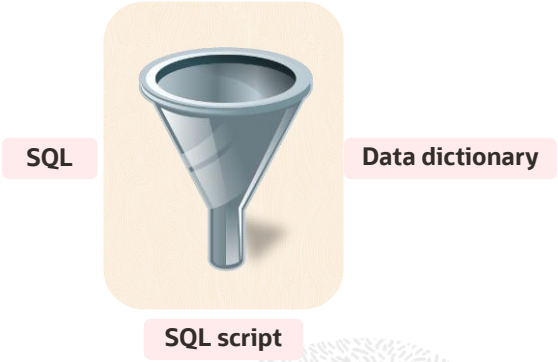
2

In this appendix, you learn how to write a SQL script to generate a SQL script.

Oracle Internal & Oracle Academy Use Only

Using SQL to Generate SQL

- SQL can be used to generate scripts in SQL.
- The data dictionary is:
  - A collection of tables and views that contain database information
  - Created and maintained by the Oracle server



3

SQL can be a powerful tool to generate other SQL statements. In most cases, this involves writing a script file. You can use SQL-from-SQL to:

- Avoid repetitive coding
- Access information from the data dictionary
- Drop or re-create database objects
- Generate dynamic predicates that contain run-time parameters

The examples used in this appendix involve selecting information from the data dictionary. The data dictionary is a collection of tables and views that contain information about the database. This collection is created and maintained by the Oracle server. All data dictionary tables are owned by the `SYS` user. Information stored in the data dictionary includes names of Oracle server users, privileges granted to users, database object names, table constraints, and audit information. There are four categories of data dictionary views. Each category has a distinct prefix that reflects its intended use.

Prefix	Description
`USER_`	Contains details of objects owned by the user
`ALL_`	Contains details of objects to which the user has been granted access rights, in addition to objects owned by the user
`DBA_`	Contains details of users with DBA privileges to access any object in the database
`V$_`	Stores information about database server performance and locking; available only to the DBA

Oracle Internal & Oracle Academy Use Only

Creating a Basic Script

```
SELECT 'CREATE TABLE ' || table_name ||
       '_test ' || 'AS SELECT * FROM '
       || table_name ||' WHERE 1=2;'
       AS "Create Table Script"
FROM   user_tables;
```

	Create Table Script
1	CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;
2	CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
3	CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
4	CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
5	CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
6	CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;

4

The example in the slide produces a report with `CREATE TABLE` statements from every table you own. Each `CREATE TABLE` statement produced in the report includes the syntax to create a table using the table name with a suffix of `_test` and having only the structure of the corresponding existing table. The old table name is obtained from the `TABLE_NAME` column of the data dictionary view `USER_TABLES`.

The next step is to enhance the report to automate the process.

Note: You can query the data dictionary tables to view various database objects that you own. The data dictionary views frequently used include:

- `USER_TABLES`: Displays description of the user's own tables
- `USER_OBJECTS`: Displays all the objects owned by the user
- `USER_TAB_PRIVS_MADE`: Displays all grants on objects owned by the user
- `USER_COL_PRIVS_MADE`: Displays all grants on columns of objects owned by the user

Oracle Internal & Oracle Academy Use Only

Controlling the Environment

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SQL statement

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

Set system variables to appropriate values.

Set system variables back to the default value.

5

To execute the SQL statements that are generated, you must capture them in a file that can then be run. You must also plan to clean up the output that is generated and make sure that you suppress elements such as headings, feedback messages, top titles, and so on. In SQL Developer, you can save these statements to a script. To save the contents of the Enter SQL Statement box, click the Save icon or select Save from the File menu. Alternatively, you can right-click in the Enter SQL Statement box and select the Save File option from the drop-down menu.

Note: Some of the SQL*Plus statements are not supported by SQL Worksheet. For the complete list of SQL*Plus statements that are supported, and not supported by SQL Worksheet, refer to the topic titled *SQL*Plus Statements Supported and Not Supported in SQL Worksheet* in SQL Developer Online Help.

Oracle Internal & Oracle Academy Use Only

The Complete Picture

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM    user_objects
WHERE   object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

6

The output of the command in the slide is saved into a file called `dropem.sql` in SQL Developer. To save the output into a file in SQL Developer, you use the Save File option under the Script Output pane. The `dropem.sql` file contains the following data. This file can now be started from SQL Developer by locating the script file, loading it, and executing it.

	'DROPTABLE'||OBJECT_NAME||';'
1	DROP TABLE REGIONS;
2	DROP TABLE COUNTRIES;
3	DROP TABLE LOCATIONS;
4	DROP TABLE DEPARTMENTS;
5	DROP TABLE JOBS;
6	DROP TABLE EMPLOYEES;
7	DROP TABLE JOB_HISTORY;
8	DROP TABLE JOB_GRADES;

Oracle Internal & Oracle Academy Use Only

Dumping the Contents of a Table to a File

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0


SELECT
 'INSERT INTO departments_test VALUES
  (' || department_id || ', ''' || department_name ||
   ''', ''' || location_id || ''');'
  AS "Insert Statements Script"
FROM   departments
/


SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

7

Sometimes, it is useful to have the values for the rows of a table in a text file in the format of an `INSERT INTO VALUES` statement. This script can be run to populate the table in case the table has been dropped accidentally.

The example in the slide produces `INSERT` statements for the `DEPARTMENTS_TEST` table, captured in the `data.sql` file using the Save File option in SQL Developer.

The contents of the `data.sql` script file are as follows:

```
INSERT INTO departments_test VALUES
    (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
    (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
    (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
    (60, 'IT', 1400);
...
```

Oracle Internal & Oracle Academy Use Only

Dumping the Contents of a Table to a File

Source	Result
'''X'''	'X'
''''	'
''''||department_name||''''	'Administration'
''', '''	','
'');'	');

8

You may have noticed the large number of single quotation marks on the previous slide. A set of four single quotation marks produces one single quotation mark in the final statement. Also remember that character and date values must be enclosed within quotation marks.

Within a string, to display one quotation mark, you need to prefix it with another single quotation mark. For example, in the fifth example in the slide, the surrounding quotation marks are for the entire string. The second quotation mark acts as a prefix to display the third quotation mark. Thus, the result is a single quotation mark followed by the parenthesis, followed by the semicolon.

Oracle Internal & Oracle Academy Use Only

Generating a Dynamic Predicate

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&&deptno', null,
DECODE ('&&hiredate', null, ' ',
'WHERE hire_date=TO_DATE('''||'&&hiredate'',''DD-MON-YYYY'')'),
DECODE ('&&hiredate', null,
'WHERE department_id = ' || '&&deptno',
'WHERE department_id = ' || '&&deptno' ||
' AND hire_date = TO_DATE('''||'&&hiredate'',''DD-MON-YYYY'')'))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```

9

The example in the slide generates a SELECT statement that retrieves data of all employees in a department who were hired on a specific day. The script generates the WHERE clause dynamically.

Note: After the user variable is in place, you must use the UNDEFINE command to delete it.

The first SELECT statement prompts you to enter the department number. If you do not enter any department number, the department number is treated as null by the DECODE function and the user is then prompted for the hire date. If you do not enter any hire date, the hire date is treated as null by the DECODE function and the dynamic WHERE clause that is generated is also a null, which causes the second SELECT statement to retrieve all the rows from the EMPLOYEES table.

Note: The NEW_V[ALUE] variable specifies a variable to hold a column value. You can reference the variable in TTITLE commands. Use NEW_VALUE to display column values or the date in the top title. You must include the column in a BREAK command with the SKIP PAGE action. The variable name cannot contain a pound sign (#). NEW_VALUE is useful for master/detail reports in which there is a new master record for each page.

Oracle Internal & Oracle Academy Use Only

Note: Here, the hire date must be entered in the `DD-MON-YYYY` format.

The `SELECT` statement in the slide can be interpreted as follows:

```
IF   (<<deptno>> is not entered)  THEN
     IF  (<<hiredate>> is not entered)  THEN
           return empty string
     ELSE
           return the string 'WHERE hire_date =
 TO_DATE('<<hiredate>>', 'DD-MON-YYYY')'
 ELSE
        IF (<<hiredate>> is not entered)  THEN
              return the string 'WHERE department_id = <<deptno>>
 entered'
        ELSE
              return the string 'WHERE department_id = <<deptno>>
 entered
                                                AND hire_date = TO_DATE('
 <<hiredate>>', 'DD-MON-YYYY')'
END IF
```

The returned string becomes the value of the `DYN_WHERE_CLAUSE` variable, which will be used in the second `SELECT` statement.

Note: Use SQL*Plus for these examples.

When the first example in the slide is executed, the user is prompted for the values for `DEPTNO` and `HIREDATE`:

Enter the values of `DEPTNO` and `HIREDATE`: 10 and `17-SEP-2007`

The following value for `MY_COL` is generated:

```
    MY_COL
1 WHERE department_id = 10 AND hire_date = TO_DATE('17-SEP-2007','DD-MON-YYYY')
```

When the second example in the slide is executed, the following output is generated:

```
LAST_NAME
-------------------------
Whalen
```

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to:

- Create a basic SQL script
- Capture the output in a file
- Dump the contents of a table to a file
- Generate a dynamic predicate

11

SQL can be used to generate SQL scripts. These scripts can be used to avoid repetitive coding, drop or re-create objects, get help from the data dictionary, and generate dynamic predicates that contain run-time parameters.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only

ORACLE

H

Oracle Database Architectural Components

0

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

- List the major database architectural components
- Describe the background processes
- Explain the memory structures
- Correlate the logical and physical storage structures

2

This appendix provides an overview of the Oracle Database architecture. You learn about the physical and logical structures and various components of Oracle Database and their functions.

Oracle Internal & Oracle Academy Use Only

Oracle Database Architecture: Overview

The Oracle Relational Database Management System (RDBMS) is a database management system that provides an open, comprehensive, integrated approach to information management.

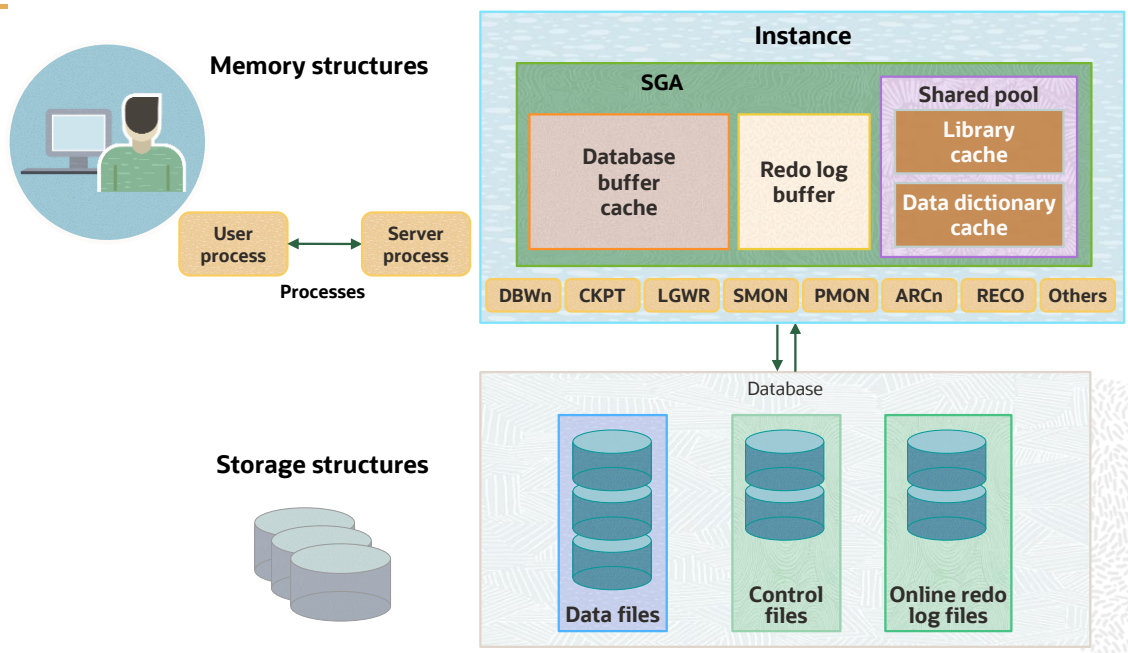
3

A database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information.

An Oracle database reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. This is accomplished while delivering high performance. At the same time, it prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Internal & Oracle Academy Use Only

Oracle Database Server Structures



4

Oracle Database consists of two main components—the instance and the database.

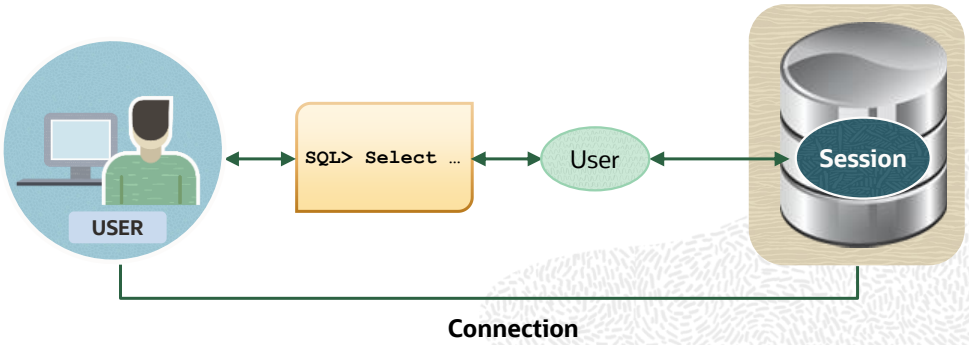
- The instance consists of the System Global Area (SGA), which is a collection of memory structures, and the background processes that perform tasks within the database. Every time an instance is started, the SGA is allocated, and the background processes are started.
- The database consists of both physical structures and logical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures. The physical storage structures include:
  - The control files where the database configuration is stored
  - The redo log files that have information required for database recovery
  - The data files where all data is stored

An Oracle instance uses memory structures and processes to manage and access the database storage structures. All memory structures exist in the main memory of the computers that constitute the database server. Processes are jobs that work in the memory of these computers. A process is defined as a “thread of control” or a mechanism in an operating system that can run a series of steps.

Oracle Internal & Oracle Academy Use Only

Connecting to the Database

- Connection: A communication pathway between a user process and a database instance
- Session: A specific connection of a user to a database instance through a user process



5

To access information in the database, the user needs to connect to the database using a tool (such as SQL*Plus). After the user establishes connection, a session is created for the user. Connection and session are closely related to user process but are very different in meaning.

A connection is a communication pathway between a user process and an Oracle Database instance. A communication pathway is established by using available interprocess communication mechanisms or network software (when different computers run the database application and Oracle Database and communicate through a network).

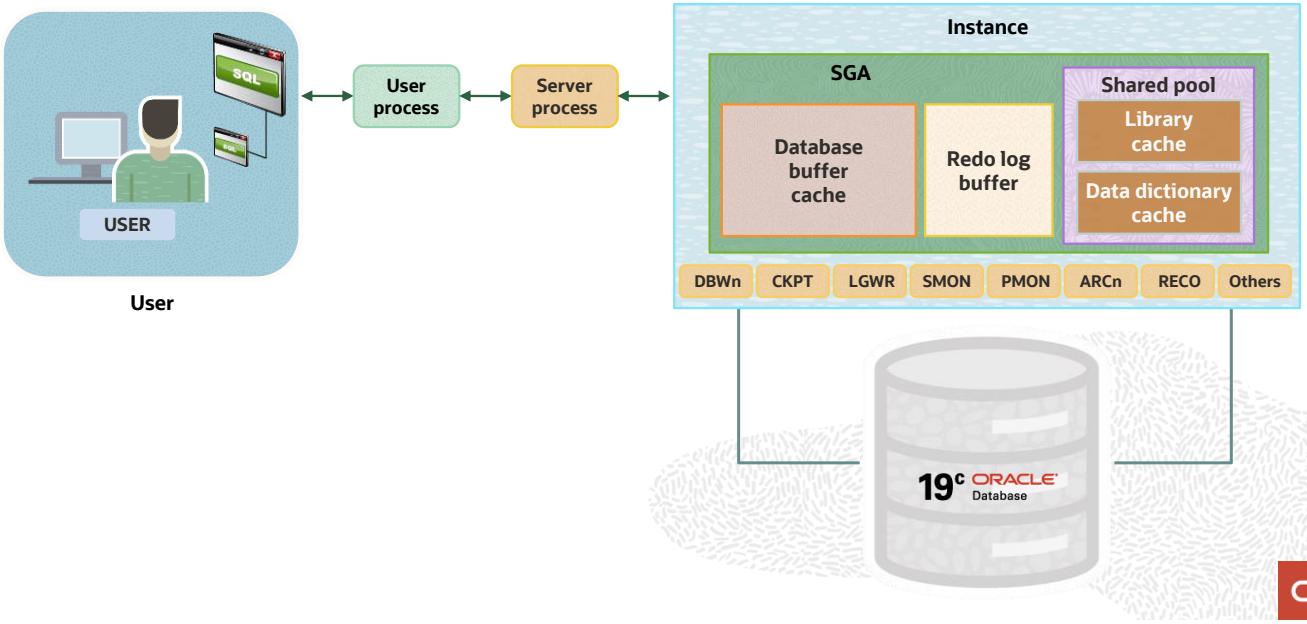
A session represents the state of a current user login to the database instance. For example, when a user starts SQL*Plus, the user must provide a valid username and password, and then a session is established for that user. A session lasts from the time the user connects until the time the user disconnects or exits the database application.

In the case of a dedicated connection, the session is serviced by a permanent dedicated process. In the case of a shared connection, the session is serviced by an available server process selected from a pool, either by the middle tier or by Oracle shared server architecture.

Multiple sessions can be created and exist concurrently for a single Oracle Database user using the same username, but through different applications, or multiple invocations of the same application.

Oracle Internal & Oracle Academy Use Only

Interacting with an Oracle Database



6

The following example describes Oracle Database operations at the most basic level. It illustrates an Oracle Database configuration where the user and the associated server process are on separate computers, connected through a network.

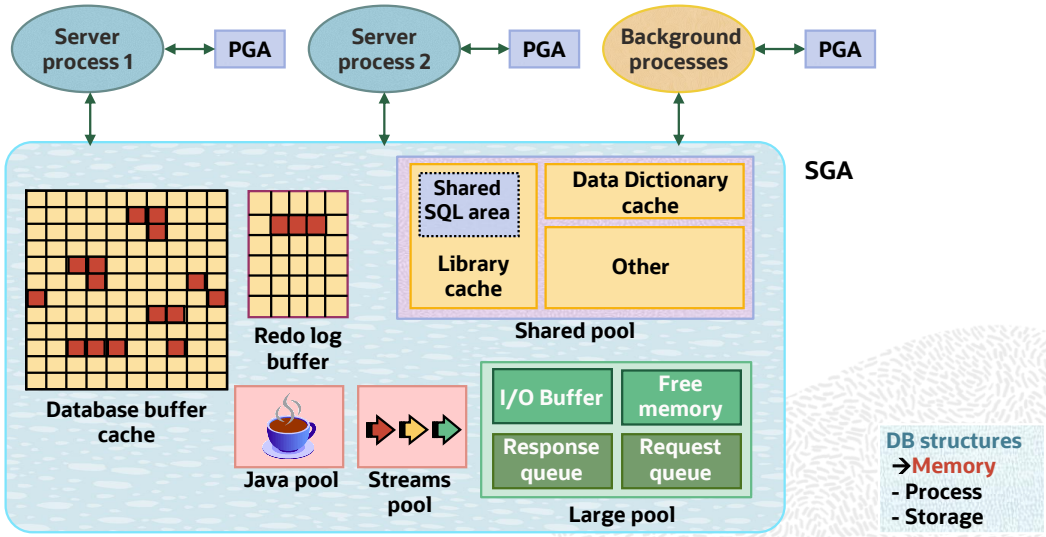
1. An instance has started on a node where Oracle Database is installed, often called the host or database server.
2. A user starts an application spawning a user process. The application attempts to establish a connection to the server. (The connection may be local, client server, or a three-tier connection from a middle tier.)
3. The server runs a listener that has the appropriate Oracle Net Services handler. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.
4. The user runs a DML-type SQL statement and commits the transaction. For example, the user changes the address of a customer in a table and commits the change.
5. The server process receives the statement and checks the shared pool (an SGA component) for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, the server process checks the user's access privileges to the requested data, and the existing shared SQL area is used to process the statement. If not, a new shared SQL area is allocated for the statement, so it can be parsed and processed.

Oracle Internal & Oracle Academy Use Only

6. The server process retrieves any necessary data values, either from the actual data file (in which the table is stored) or from those cached in the SGA.
7. The server process modifies data in the SGA. Because the transaction is committed, the log writer process (LGWR) immediately records the transaction in the redo log file. The database writer (DBW*n*) process writes modified blocks permanently to disk when doing so is efficient.
8. If the transaction is successful, the server process sends a message across the network to the application. If it is not successful, an error message is transmitted.
9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

Oracle Internal & Oracle Academy Use Only

Oracle Memory Architecture



8

Oracle Database creates and uses memory structures for various purposes. For example, memory stores program code being run, data shared among users, and private data areas for each connected user. Two basic memory structures are associated with an instance:

- The SGA is a group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- The Program Global Areas (PGA) are memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

Oracle Internal & Oracle Academy Use Only

The SGA is the memory area that contains data and control information for the instance. The SGA includes the following data structures:

- **Database buffer cache:** Caches blocks of data retrieved from the database
- **Redo Log buffer:** Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Shared pool:** Caches various constructs that can be shared among users
- **Large pool:** Is an optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and input/output (I/O) server processes
- **Java pool:** Is used for all session-specific Java code and data within the Java Virtual Machine (JVM)
- **Streams pool:** Is used by Oracle Streams to store information required by capture and apply

When you start the instance by using Enterprise Manager or SQL*Plus, the amount of memory allocated for the SGA is displayed.

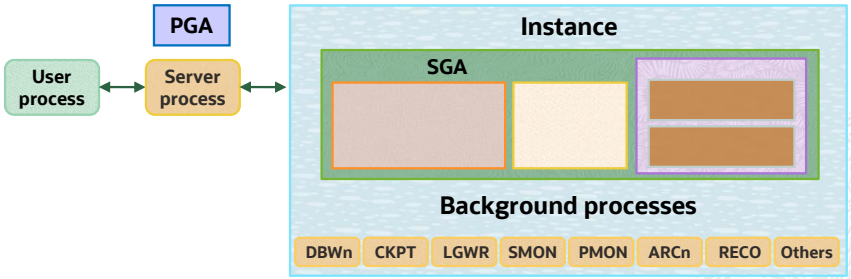
With the dynamic SGA infrastructure, the size of the database buffer cache, the shared pool, the large pool, the Java pool, and the Streams pool changes without shutting down the instance.

Oracle Database uses initialization parameters to create and configure memory structures. For example, the `SGA_TARGET` parameter specifies the total size of the SGA components. If you set `SGA_TARGET` to 0, Automatic Shared Memory Management is disabled.

Oracle Internal & Oracle Academy Use Only

Process Architecture

- User process:
  - Is started when a database user or a batch process connects to Oracle Database
- Database processes:
  - Server process: Connects to the Oracle instance and is started when a user establishes a session
  - Background processes: Are started when an Oracle instance is started



DB structures
- Memory
→Process
- Storage

10

The processes in an Oracle Database server can be categorized into two major groups:

- User processes that run the application or Oracle tool code
- Oracle Database processes that run the Oracle database server code. These include server processes and background processes.

When a user runs an application program or an Oracle tool such as SQL*Plus, Oracle Database creates a user process to run the user's application. Oracle Database also creates a server process to execute the commands issued by the user process. In addition, the Oracle server also has a set of background processes for an instance that interact with each other and with the operating system to manage the memory structures and asynchronously perform I/O to write data to disk and perform other required tasks.

The process structure varies for different Oracle Database configurations, depending on the operating system and the choice of Oracle Database options. The code for connected users can be configured as a dedicated server or a shared server.

- With a dedicated server, for each user, the database application is run by a different process (a user process) than the one that runs the Oracle server code (a dedicated server process).
- A shared server eliminates the need for a dedicated server process for each connection. A dispatcher directs multiple incoming network session requests to a pool of shared server processes. A shared server process serves any client request.

Oracle Internal & Oracle Academy Use Only

Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to the instance. In some situations when the application and Oracle Database operate on the same computer, it is possible to combine the user process and the corresponding server process into a single process to reduce system overhead. However, when the application and Oracle Database operate on different computers, a user process always communicates with Oracle Database through a separate server process.

Server processes created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application.
- Read necessary data blocks from data files on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA.
- Return results in such a way that the application can process the information.

Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses some additional Oracle Database processes called background processes. An Oracle Database instance can have many background processes.

The following background processes are required for a successful startup of the database instance:

- Database writer (DBW*n*)
- Log writer (LGWR)
- Checkpoint (CKPT)
- System monitor (SMON)
- Process monitor (PMON)

The following background processes are a few examples of optional background processes that can be started if required:

- Recoverer (RECO)
- Job queue
- Archiver (ARC*n*)
- Queue monitor (QMN*n*)

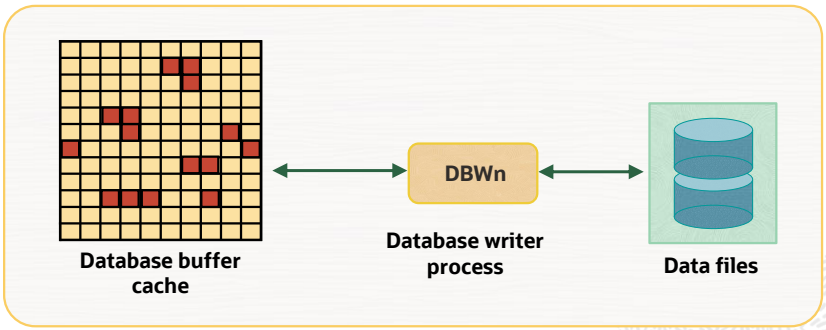
Other background processes may be found in more advanced configurations such as Real Application Clusters (RAC). See the `V$BGPROCESS` view for more information about the background processes. On many operating systems, background processes are created automatically when an instance is started.

Oracle Internal & Oracle Academy Use Only

Database Writer Process

Writes modified (dirty) buffers in the database buffer cache to disk:

- Asynchronously while performing other processing
- Periodically to advance the checkpoint



12

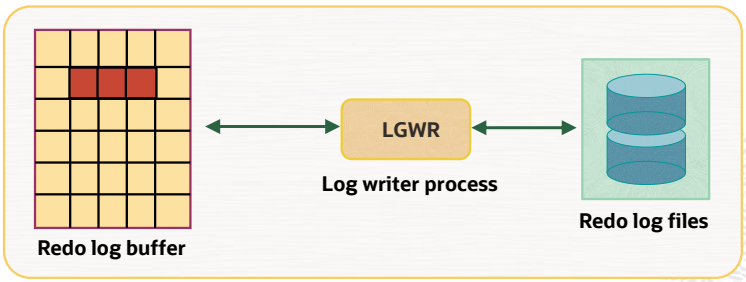
The database writer (DBW*n*) process writes the contents of buffers to data files. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes (DBW1 through DBW9 and DBWa through DBWj) to improve write performance if your system modifies data heavily. These additional DBW*n* processes are not useful on uniprocessor systems.

When a buffer in the database buffer cache is modified, it is marked “dirty” and is added to the LRUW list of dirty buffers that is kept in system change number (SCN) order, thereby matching the order of Redo corresponding to these changed buffers that is written to the Redo logs. When the number of available buffers in the buffer cache falls below an internal threshold such that server processes find it difficult to obtain available buffers, DBW*n* writes dirty buffers to the data files in the order that they were modified by following the order of the LRUW list.

Oracle Internal & Oracle Academy Use Only

Log Writer Process

- Writes the redo log buffer to a redo log file on disk
- The Log Writer (LGWR) writes:
  - When a process commits a transaction
  - When the redo log buffer is one-third full
  - Before a DBWn process writes modified buffers to disk



13

The log writer (LGWR) process is responsible for redo log buffer management by writing the redo log buffer entries to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. The LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

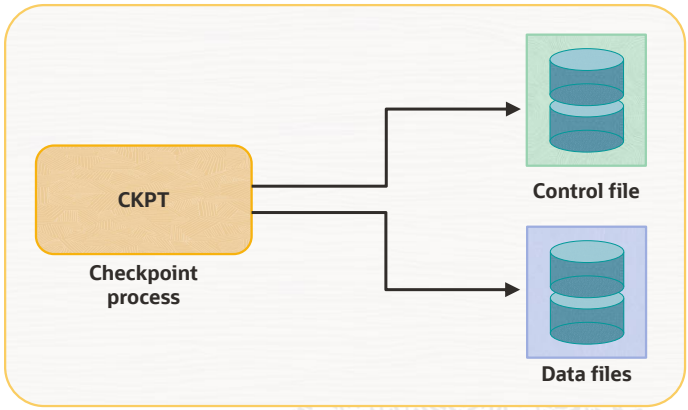
- When a user process commits a transaction
- When the redo log buffer is one-third full
- Before a DBW*n* process writes modified buffers to disk, if necessary

Oracle Internal & Oracle Academy Use Only

Checkpoint Process

Records checkpoint information in:

- The control file
- Each data file header



14

A checkpoint is a data structure that defines an SCN in the redo thread of a database. Checkpoints are recorded in the control file and each data file header, and are a crucial element of recovery.

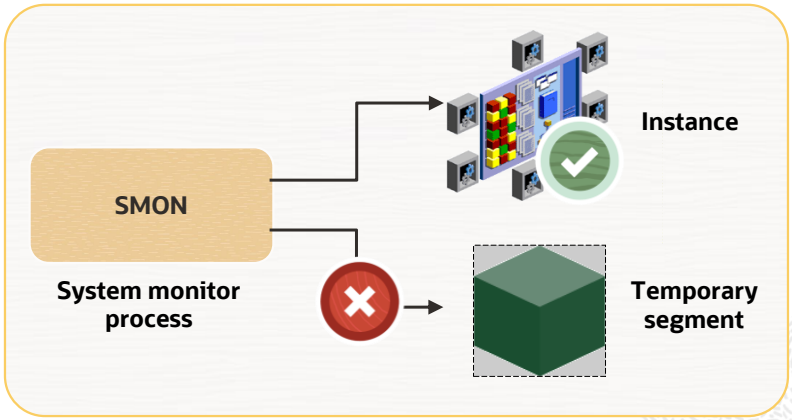
When a checkpoint occurs, Oracle Database must update the headers of all data files to record the details of the checkpoint. This is done by the CKPT process. The CKPT process does not write blocks to disk; DBW*n* always performs that work. The SCNs recorded in the file headers guarantee that all the changes made to database blocks before that SCN have been written to disk.

The statistic DBWR checkpoints displayed by the `SYSTEM_STATISTICS` monitor in Oracle Enterprise Manager indicate the number of checkpoint requests completed.

Oracle Internal & Oracle Academy Use Only

System Monitor Process

- Performs recovery at instance startup
- Cleans up unused temporary segments



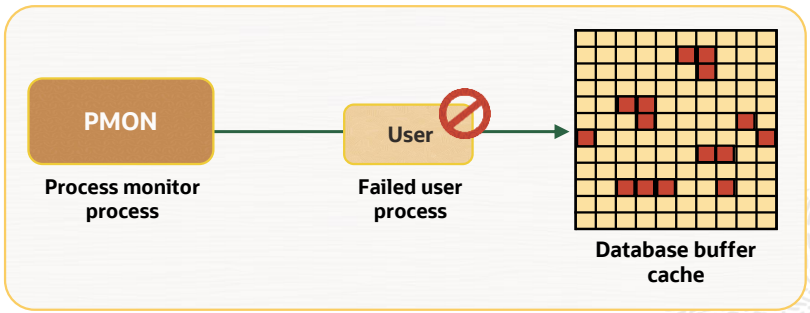
15

The system monitor (SMON) process performs recovery, if necessary, at instance startup. It is also responsible for cleaning up temporary segments that are no longer in use. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or the file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

Oracle Internal & Oracle Academy Use Only

Process Monitor Process

- Performs process recovery when a user process fails
- Monitors sessions for idle session timeout
- Dynamically registers database services with listeners



16

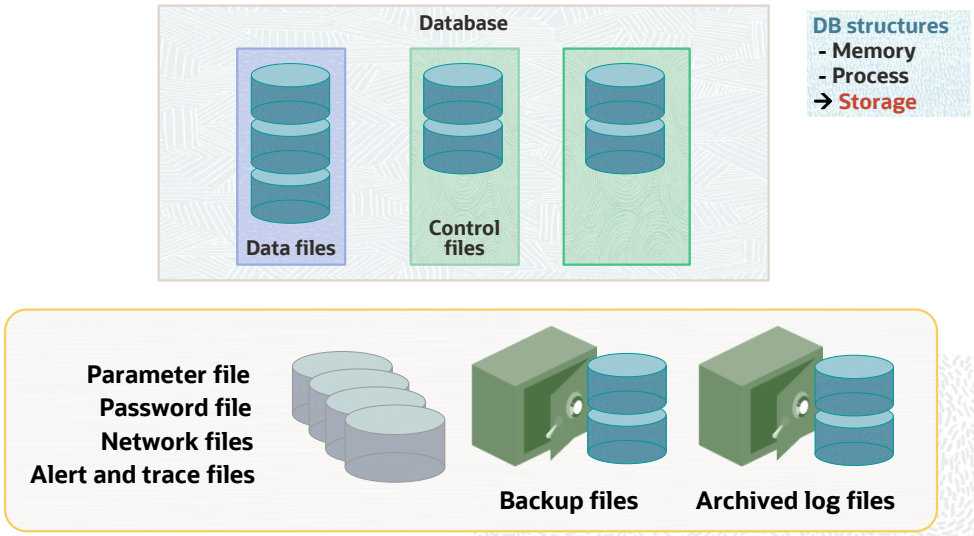
The process monitor (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the process ID from the list of active processes.

PMON periodically checks the status of dispatcher and server processes and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.

Like SMON, PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

Oracle Internal & Oracle Academy Use Only

Oracle Database Storage Architecture



17

The files that constitute an Oracle database are organized into the following:

- **Control files:** Contain data about the database itself (that is, physical database structure information). These files are critical to the database. Without them, you cannot open data files to access the data within the database.
- **Data files:** Contain the user or application data of the database, as well as metadata and the data dictionary
- **Online redo log files:** Allow for instance recovery of the database. If the database server crashes and does not lose any data files, the instance can recover the database with the information in these files.

The following additional files are important to the successful running of the database:

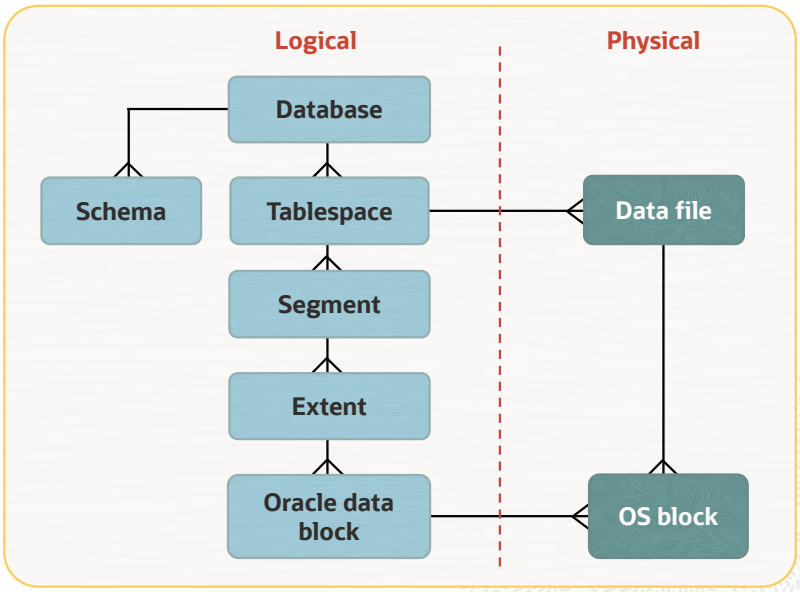
- **Backup files:** Are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- **Archived log files:** Contain an ongoing history of the data changes (redo) that are generated by the instance. Using these files and a backup of the database, you can recover a lost data file. That is, archive logs enable the recovery of restored data files.
- **Parameter file:** Is used to define how the instance is configured when it starts up
- **Password file:** Allows `sysdba/sysoper/sysasm` to connect remotely to the database and perform administrative tasks

Oracle Internal & Oracle Academy Use Only

- **Network files:** Are used for starting the database listener and store information required for user connections
- **Trace files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information is for Oracle Support Services.
- **Alert log files:** These are special trace entries. The alert log of a database is a chronological log of messages and errors. Each instance has one alert log file. Oracle recommends that you review this alert log periodically.

Oracle Internal & Oracle Academy Use Only

Logical and Physical Database Structures



19

An Oracle database has logical and physical storage structures.

Tablespaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group all of an application's objects to simplify some administrative operations. You may have a tablespace for application data and an additional one for application indexes.

Databases, Tablespaces, and Data Files

The relationship among databases, tablespaces, and data files is illustrated in the slide. Each database is logically divided into one or more tablespaces. One or more data files are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. If it is a `TEMPORARY` tablespace, instead of a data file, the tablespace has a temporary file.

Oracle Internal & Oracle Academy Use Only

Schemas

A schema is a collection of database objects that are owned by a database user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. In general, schema objects include everything that your application creates in the database.

Data Blocks

At the finest level of granularity, an Oracle database's data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on the disk. A data block size is specified for each tablespace when it is created. A database uses and allocates free database space in Oracle data blocks.

Extents

The next level of logical database space is called an extent. An extent is a specific number of contiguous data blocks (obtained in a single allocation) that are used to store specific type of information.

Segments

The level of logical database storage above an extent is called a segment. A segment is a set of extents allocated for a certain logical structure. For example, the different types of segments include:

- **Data segments:** Each nonclustered, non-indexed-organized table has a data segment with the exception of external tables, global temporary tables, and partitioned tables, where each table has one or more segments. All of the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.
- **Index segments:** Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
- **Undo segments:** One `UNDO` tablespace is created per database instance that contains numerous undo segments to temporarily store *undo* information. The information in an undo segment is used to generate read-consistent database information and, during database recovery, to roll back uncommitted transactions for users.
- **Temporary segments:** Temporary segments are created by the Oracle Database when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the instance for future use. Specify a default temporary tablespace for every user or a default temporary tablespace, which is used databasewide.

The Oracle Database dynamically allocates space. When the existing extents of a segment are full, additional extents are added. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on the disk.

Oracle Internal & Oracle Academy Use Only

Processing a SQL Statement

- Connect to an instance using:
  - The user process
  - The server process
- The Oracle server components that are used depend on the type of SQL statement.
- Some Oracle server components do not participate in SQL statement processing.

21

Not all the components of an Oracle instance are used to process SQL statements. The user and server processes are used to connect a user to an Oracle instance. These processes are not a part of the Oracle instance, but are required to process a SQL statement.

Some of the background processes, SGA structures, and database files are used to process SQL statements. Depending on the type of SQL statement, different components are used:

- Queries require additional processing to return rows to the user.
- DML statements require additional processing to log the changes made to the data.
- Commit processing ensures that the modified data in a transaction can be recovered.

Some required background processes do not directly participate in processing a SQL statement, but are used to improve performance and to recover the database. For example, the optional Archiver background process, ARC*n*, is used to ensure that a production database can be recovered.

Oracle Internal & Oracle Academy Use Only

Processing a Query

- Parse:
  - Search for an identical statement
  - Check the syntax, object names, and privileges
  - Lock the objects used during parse
  - Create and store the execution plan
- Execute: Identify the rows selected
- Fetch: Return the rows to the user process

22

Queries are different from other types of SQL statements because, if successful, they return data as results. Other statements simply return success or failure, whereas a query can return one row or thousands of rows.

There are three main stages in the processing of a query:

- Parse
- Execute
- Fetch

During the parse stage, the SQL statement is passed from the user process to the server process, and a parsed representation of the SQL statement is loaded into a shared SQL area.

During parse, the server process performs the following functions:

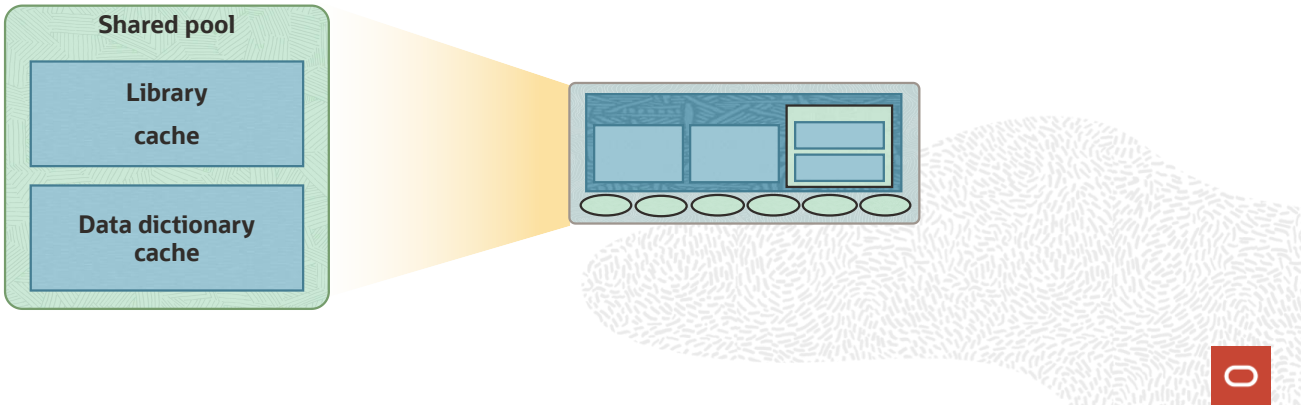
- Searches for an existing copy of the SQL statement in the shared pool
- Validates the SQL statement by checking its syntax
- Performs data dictionary lookups to validate table and column definitions

The execute stage executes the statement using the best optimizer approach and the fetch stage retrieves the rows back to the user.

Oracle Internal & Oracle Academy Use Only

Shared Pool

- The library cache contains the SQL statement text, parsed code, and execution plan.
- The data dictionary cache contains table, column, and other object definitions and privileges.
- The shared pool is sized by `SHARED_POOL_SIZE`.



23

During the parse stage, the server process uses the area in the SGA known as the shared pool to compile the SQL statement. The shared pool has two primary components:

- Library cache
- Data dictionary cache

Library Cache

The library cache stores information about the most recently used SQL statements in a memory structure called a shared SQL area. The shared SQL area contains:

- The text of the SQL statement
- The parse tree, which is a compiled version of the statement
- The execution plan, with steps to be taken when executing the statement

The optimizer is the function in the Oracle server that determines the optimal execution plan.

If a SQL statement is reexecuted and a shared SQL area already contains the execution plan for the statement, the server process does not need to parse the statement. The library cache improves the performance of applications that reuse SQL statements by reducing parse time and memory requirements. If the SQL statement is not reused, it is eventually aged out of the library cache.

Oracle Internal & Oracle Academy Use Only

Data Dictionary Cache

The data dictionary cache, also known as the dictionary cache or row cache, is a collection of the most recently used definitions in the database. It includes information about database files, tables, indexes, columns, users, privileges, and other database objects.

During the parse phase, the server process looks for the information in the dictionary cache to resolve the object names specified in the SQL statement and to validate the access privileges. If necessary, the server process initiates the loading of this information from the data files.

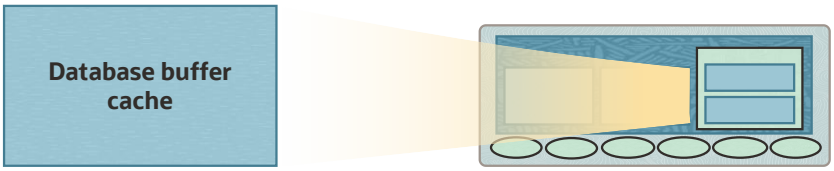
Sizing the Shared Pool

The size of the shared pool is specified by the `SHARED_POOL_SIZE` initialization parameter.

Oracle Internal & Oracle Academy Use Only

Database Buffer Cache

- The database buffer cache stores the most recently used blocks.
- The size of a buffer is based on `DB_BLOCK_SIZE`.
- The number of buffers is defined by `DB_BLOCK_BUFFERS`.



25

When a query is processed, the server process looks in the database buffer cache for any blocks it needs. If the block is not found in the database buffer cache, the server process reads the block from the data file and places a copy in the buffer cache. Because subsequent requests for the same block may find the block in memory, the requests may not require physical reads. The Oracle server uses a least recently used algorithm to age out buffers that have not been accessed recently to make room for new blocks in the buffer cache.

Sizing the Database Buffer Cache

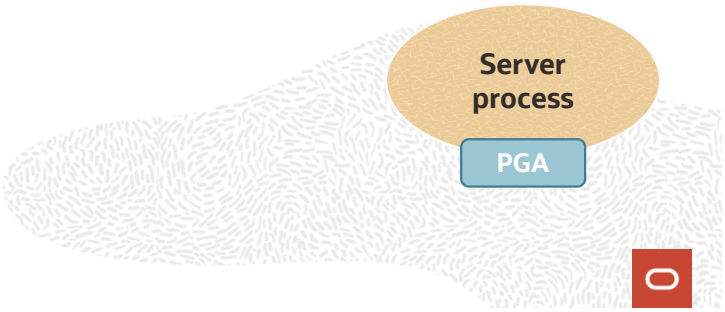
The size of each buffer in the buffer cache is equal to the size of an Oracle block, and it is specified by the `DB_BLOCK_SIZE` parameter. The number of buffers is equal to the value of the `DB_BLOCK_BUFFERS` parameter.

Oracle Internal & Oracle Academy Use Only

Program Global Area

A Program Global Area (PGA):

- Is not shared
- Is writable only by the server process
- Contains:
  - Sort area
  - Session information
  - Cursor state
  - Stack space



26

A Program Global Area (PGA) is a memory region that contains data and control information for a server process. It is a nonshared memory created by Oracle when a server process is started. Access to it is exclusive to that server process and is read and written only by the Oracle server code acting on behalf of it. The PGA memory allocated by each server process attached to an Oracle instance is referred to as the aggregated PGA memory allocated by the instance.

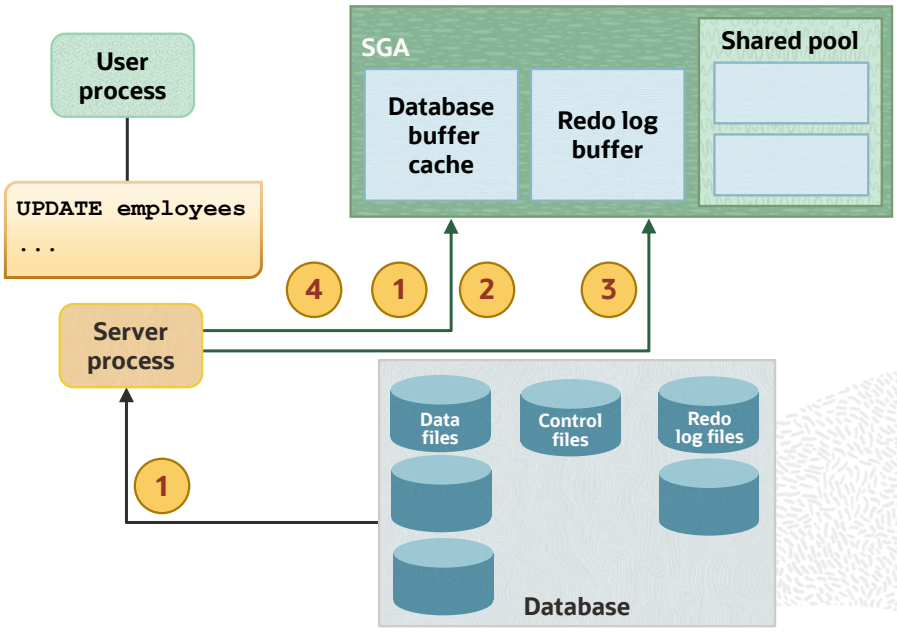
In a dedicated server configuration, the PGA of the server includes the following components:

- **Sort area:** Is used for any sorts that may be required to process the SQL statement
- **Session information:** Includes user privileges and performance statistics for the session
- **Cursor state:** Indicates the stage in the processing of the SQL statements that are currently used by the session
- **Stack space:** Contains other session variables

The PGA is allocated when a process is created and deallocated when the process is terminated.

Oracle Internal & Oracle Academy Use Only

Processing a DML Statement



27

A data manipulation language (DML statement requires only two phases of processing:

- Parse is the same as the parse phase used for processing a query.
- Execute requires additional processing to make data changes.

DML Execute Phase

To execute a DML statement:

- If the data and rollback blocks are not already in the buffer cache, the server process reads them from the data files into the buffer cache.
- The server process places locks on the rows that are to be modified.
- In the redo log buffer, the server process records the changes to be made to the rollback and data blocks.
- The rollback block changes record the values of the data before it is modified. The rollback block is used to store the “before image” of the data, so that the DML statements can be rolled back if necessary.
- The data block changes record the new values of the data.

Oracle Internal & Oracle Academy Use Only

The server process records the “before image” to the rollback block and updates the data block. Both of these changes are done in the database buffer cache. Any changed blocks in the buffer cache are marked as dirty buffers (that is, buffers that are not the same as the corresponding blocks on the disk).

The processing of a `DELETE` or `INSERT` command uses similar steps. The “before image” for a `DELETE` contains the column values in the deleted row, and the “before image” of an `INSERT` contains the row location information.

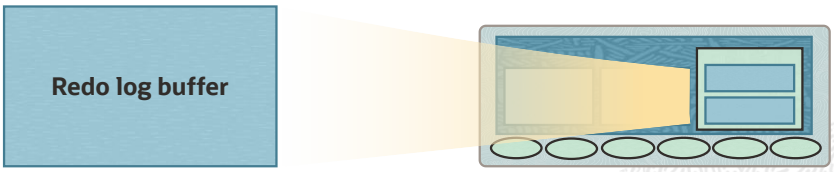
Because the changes made to the blocks are only recorded in memory structures and are not written immediately to disk, a computer failure that causes the loss of the SGA can also lose these changes.

Oracle Internal & Oracle Academy Use Only

Redo Log Buffer

Redo Log Buffer:

- Has its size defined by `LOG_BUFFER`
- Records changes made through the instance
- Is used sequentially
- Is a circular buffer



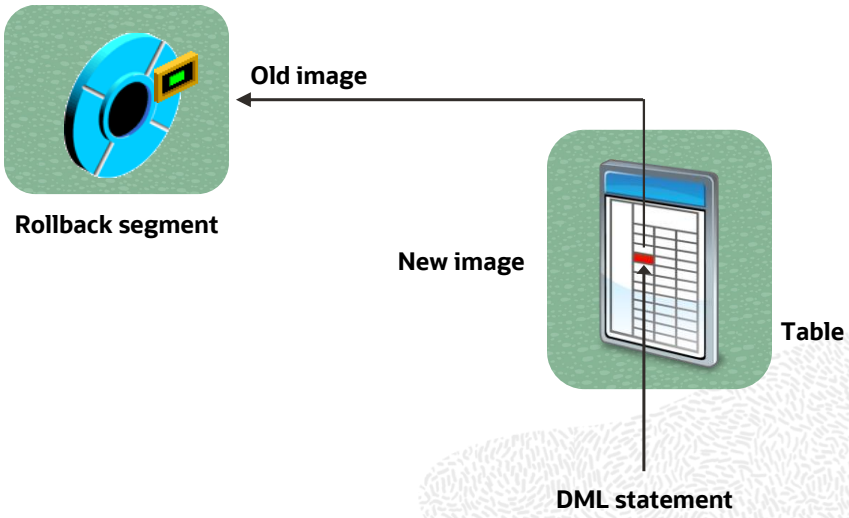
29

The server process records most of the changes made to data file blocks in the redo log buffer, which is a part of the SGA. The redo log buffer has the following characteristics:

- Its size in bytes is defined by the `LOG_BUFFER` parameter.
- It records the block that is changed, the location of the change, and the new value in a redo entry. A redo entry makes no distinction between the types of block that is changed; it only records which bytes are changed in the block.
- The redo log buffer is used sequentially, and changes made by one transaction may be interleaved with changes made by other transactions.
- It is a circular buffer that is reused after it is filled, but only after all the old redo entries are recorded in the redo log files.

Oracle Internal & Oracle Academy Use Only

Rollback Segment



30

Before making a change, the server process saves the old data value in a rollback segment. This “before image” is used to:

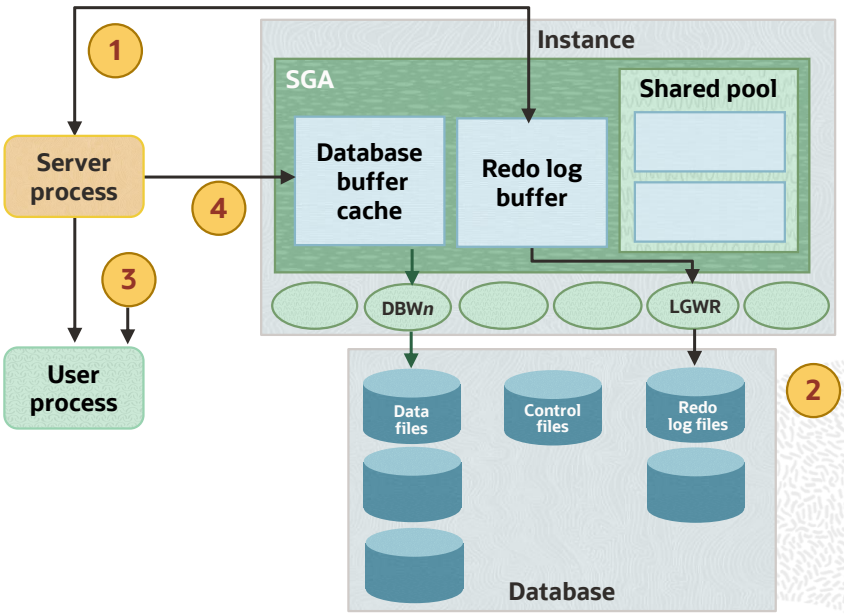
- Undo the changes if the transaction is rolled back
- Provide read consistency by ensuring that other transactions do not see uncommitted changes made by the DML statement
- Recover the database to a consistent state in case of failures

Rollback segments, such as tables and indexes, exist in data files, and rollback blocks are brought into the database buffer cache as required. Rollback segments are created by the DBA.

Changes to rollback segments are recorded in the redo log buffer.

Oracle Internal & Oracle Academy Use Only

COMMIT Processing



31

The Oracle server uses a fast COMMIT mechanism that guarantees that the committed changes can be recovered in case of instance failure.

System Change Number

Whenever a transaction commits, the Oracle server assigns a commit SCN to the transaction. The SCN is monotonically incremented and is unique within the database. It is used by the Oracle server as an internal time stamp to synchronize data and to provide read consistency when data is retrieved from the data files. Using the SCN enables the Oracle server to perform consistency checks without depending on the date and time of the operating system.

Steps in Processing COMMITs

When a COMMIT is issued, the following steps are performed:

1. The server process places a commit record, along with the SCN, in the redo log buffer.
2. LGWR performs a contiguous write of all the redo log buffer entries up to and including the commit record to the redo log files. After this point, the Oracle server can guarantee that the changes will not be lost even if there is an instance failure.

Oracle Internal & Oracle Academy Use Only

3. The user is informed that the `COMMIT` is complete.
4. The server process records information to indicate that the transaction is complete and that resource locks can be released.

Flushing of the dirty buffers to the data file is performed independently by DBW0 and can occur either before or after the commit.

Advantages of the Fast `COMMIT`

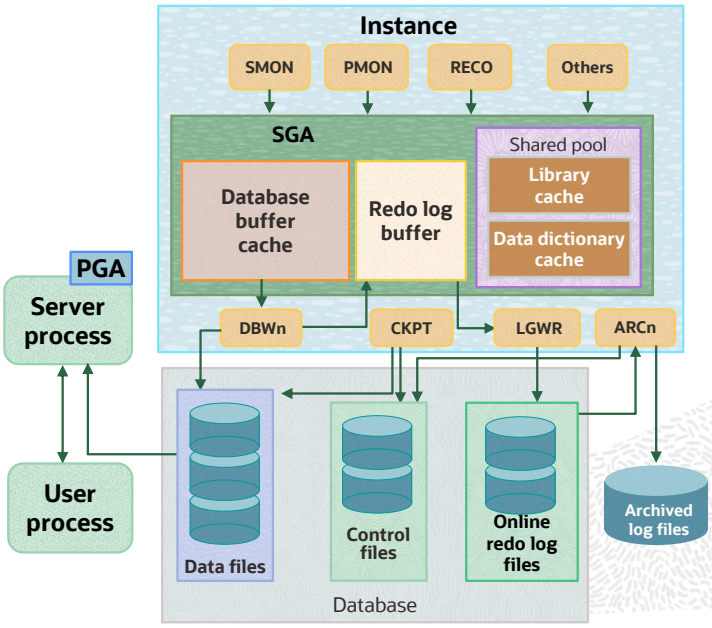
The fast `COMMIT` mechanism ensures data recovery by writing changes to the redo log buffer instead of the data files. It has the following advantages:

- Sequential writes to the log files are faster than writing to different blocks in the data file.
- Only the minimal information that is necessary to record changes is written to the log files; writing to the data files would require whole blocks of data to be written.
- If multiple transactions request to commit at the same time, the instance piggybacks redo log records into a single write.
- Unless the redo log buffer is particularly full, only one synchronous write is required per transaction. If piggybacking occurs, there can be less than one synchronous write per transaction.
- Because the redo log buffer may be flushed before the `COMMIT`, the size of the transaction does not affect the amount of time needed for an actual `COMMIT` operation.

Note: Rolling back a transaction does not trigger LGWR to write to disk. The Oracle server always rolls back uncommitted changes when recovering from failures. If there is a failure after a rollback, before the rollback entries are recorded on disk, the absence of a commit record is sufficient to ensure that the changes made by the transaction are rolled back.

Oracle Internal & Oracle Academy Use Only

Summary of the Oracle Database Architecture



33

An Oracle database comprises an instance and its associated database:

- An instance comprises the SGA and the background processes
  - **SGA:** Database buffer cache, redo log buffer, shared pool, and so on
  - **Background processes:** SMON, PMON, DBW*n*, CKPT, LGWR, and so on
- A database comprises storage structures:
  - **Logical:** Tablespaces, schemas, segments, extents, and Oracle block
  - **Physical:** Data files, control files, redo log files

When a user accesses the Oracle database through an application, a server process communicates with the instance on behalf of the user process.

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to:

- List the major database architectural components
- Describe the background processes
- Explain the memory structures
- Correlate the logical and physical storage structures

34

Oracle Internal & Oracle Academy Use Only

ORACLE

1

Regular Expression Support

0

Oracle Internal & Oracle Academy Use Only

Objectives

After completing this appendix, you should be able to:

- List the benefits of using regular expressions
- Use regular expressions to search for, match, and replace strings

2

In this appendix, you learn to use the regular expression support feature. Regular expression support is available in both SQL and PL/SQL.

Oracle Internal & Oracle Academy Use Only

What Are Regular Expressions?

- You use regular expressions to search for (and manipulate) simple and complex patterns in string data by using standard syntax conventions.
- You use a set of SQL functions and conditions to search for and manipulate strings in SQL and PL/SQL.
- You specify a regular expression by using:
  - Metacharacters
  - Literals

3

Oracle Database provides support for regular expressions. The implementation complies with the Portable Operating System for UNIX (POSIX) standard, controlled by the Institute of Electrical and Electronics Engineers (IEEE), for ASCII data-matching semantics and syntax. Oracle's multilingual capabilities extend the matching capabilities of the operators beyond the POSIX standard. Regular expressions are a method of describing both simple and complex patterns for searching and manipulating.

String manipulation and searching contribute to a large percentage of the logic within a web-based application. Usage ranges from the simple, such as finding the words "San Francisco" in a specified text, to the complex task of extracting all URLs from the text and the more complex task of finding all words whose every second character is a vowel.

When coupled with native SQL, the use of regular expressions allows for very powerful search and manipulation operations on any data stored in an Oracle database. You can use this feature to easily solve problems that would otherwise involve complex programming.

You specify a regular expression by using:

- Metacharacters, which are operators that specify the search algorithms
- Literals, which are the characters for which you are searching

Oracle Internal & Oracle Academy Use Only

Benefits of Using Regular Expressions

Regular expressions enable you to implement complex match logic in the database with the following benefits:

- By centralizing match logic in Oracle Database, you avoid intensive string processing of SQL result sets by middle-tier applications.
- Using server-side regular expressions to enforce constraints, you eliminate the need to code data validation logic on the client.
- The built-in SQL and PL/SQL regular expression functions and conditions make string manipulations more powerful and easier than in previous releases of Oracle Database.

4

Regular expressions are a powerful text-processing component of programming languages such as PERL and Java. For example, a PERL script can process each HTML file in a directory, read its contents into a scalar variable as a single string, and then use regular expressions to search for URLs in the string. One reason for many developers writing in PERL is that it has a robust pattern-matching functionality. Oracle's support of regular expressions enables developers to implement complex match logic in the database. Regular expressions were introduced in Oracle Database 10*g*.

Oracle Internal & Oracle Academy Use Only

Using the Regular Expressions Functions and Conditions in SQL and PL/SQL

Function or Condition Name	Description
REGEXP_LIKE	Is similar to the LIKE operator, but performs regular expression matching instead of simple pattern matching (condition)
REGEXP_REPLACE	Searches for a regular expression pattern and replaces it with a replacement string
REGEXP_INSTR	Searches a string for a regular expression pattern and returns the position where the match is found
REGEXP_SUBSTR	Searches for a regular expression pattern within a given string and extracts the matched substring
REGEXP_COUNT	Returns the number of times a pattern match is found in an input sting

5

Oracle Database provides a set of SQL functions that you use to search and manipulate strings by using regular expressions. You use these functions on a text literal, bind variable, or any column that holds character data, such as CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2, and VARCHAR2 (but not LONG). A regular expression must be enclosed within single quotation marks. This ensures that the entire expression is interpreted by the SQL function and can improve the readability of your code.

- REGEXP_LIKE: This condition searches a character column for a pattern. Use this condition in the WHERE clause of a query to return rows matching the regular expression that you specify.
- REGEXP_REPLACE: This function searches for a pattern in a character column and replaces each occurrence of that pattern with the pattern that you specify.
- REGEXP_INSTR: This function searches a string for a given occurrence of a regular expression pattern. You specify which occurrence you want to find and the start position to search from. This function returns an integer indicating the position in the string where the match is found.
- REGEXP_SUBSTR: This function returns the actual substring matching the regular expression pattern that you specify.
- REGEXP_COUNT: This function returns the number of times a pattern match is found in the input string.

Oracle Internal & Oracle Academy Use Only

What Are Metacharacters?

- Metacharacters are special characters that have a special meaning.
- You can use several predefined metacharacter symbols in the pattern matching.
- For example, the `^(f|ht)tps?:$` regular expression searches for the following from the beginning of the string:
  - The literals `f` or `ht`
  - The `t` literal
  - The `p` literal, optionally followed by the `s` literal
  - The colon ":" literal at the end of the string

6

Metacharacters are special characters that have a special meaning, such as a wildcard, a repeating character, a nonmatching character, or a range of characters.

The regular expression in the slide matches the `http:`, `https:`, `ftp:`, and `ftps:` strings.

Oracle Internal & Oracle Academy Use Only

Using Metacharacters with Regular Expressions

Syntax	Description
.	Matches any character in the supported character set, except NULL
+	Matches one or more occurrences
?	Matches zero or one occurrence
*	Matches zero or more occurrences of the preceding subexpression
{m}	Matches exactly m occurrences of the preceding expression
{m, }	Matches at least m occurrences of the preceding subexpression
{m,n}	Matches at least m, but not more than n, occurrences of the preceding subexpression
[...]	Matches any single character in the list within the brackets
|	Matches one of the alternatives
(...)	Treats the enclosed expression within the parentheses as a unit. The subexpression can be a string of literals or a complex expression containing operators.

7

Any character, " . " : **a.b** matches the strings **abb**, **acb**, and **adb**, but not **acc**.

One or more, " + " : **a+** matches the strings **a**, **aa**, and **aaa**, but does not match **bbb**.

Zero or one, " ? ": **ab?c** matches the strings **abc** and **ac**, but does not match **abbc**.

Zero or more, " * ": **ab*c** matches the strings **ac**, **abc**, and **abbc**, but does not match **abb**.

Exact count, " {m} ": **a{3}** matches the strings **aaa**, but does not match **aa**.

At least count, " {m,} " : **a{3,}** matches the strings **aaa** and **aaaa**, but not **aa**.

Between count, " {m,n} " : **a{3,5}** matches the strings **aaa**, **aaaa**, and **aaaaa**, but not **aa**.

Matching character list, " [...] ": **[abc]** matches the first character in the strings **all**, **bill**, and **cold**, but does not match any characters in **doll**.

Or, " | ": **a|b** matches character **a** or character **b**.

Subexpression, " (...) ": **(abc)?def** matches the optional string **abc**, followed by **def**. The expression matches **abcdefghi** and **def**, but does not match **ghi.** The subexpression can be a string of literals or a complex expression containing operators.

Oracle Internal & Oracle Academy Use Only

Using Metacharacters with Regular Expressions

Syntax	Description
^	Matches the beginning of a string
$	Matches the end of a string
\	Treats the subsequent metacharacter in the expression as a literal
\n	Matches the nth (1–9) preceding subexpression of whatever is grouped within parentheses. The parentheses cause an expression to be remembered; a backreference refers to it.
\d	A digit character
[:class:]	Matches any character belonging to the specified POSIX character class
[^:class:]	Matches any single character not in the list within the brackets

8

Beginning/end of line anchor, " ^ " and "$":** **^def** matches **def** in the string **defghi** but does not match **def** in **abcdef**. **def$ matches **def** in the string **abcdef** but does not match **def** in the string **defghi**.

Escape character " \ ": **\+** searches for a **+**. It matches the plus character in the string **abc+def**, but does not match **Abcdef**.

Backreference, " \n ": **(abc|def)xy\1** matches the strings **abcxyabc** and **defxydef**, but does not match **abcxydef** or **abcxy**. A backreference enables you to search for a repeated string without knowing the actual string ahead of time. For example, the expression **^(.*)\1$** matches a line consisting of two adjacent instances of the same string.

Digit character, "\d": The expression ^\[\d{3}\] \d{3}-\d{4}$ matches [650] 555-1212 but does not match 650-555-1212.

Character class, " [:class:] ": **[[:upper:]]+** searches for one or more consecutive uppercase characters. This matches **DEF** in the string **abcDEFghi** but does not match the string **abcdefghi**.

Nonmatching character list (or class), " [^...] ": **[^abc]** matches the character **d** in the string **abcdef**, but not **a**, **b**, or **c**.

Oracle Internal & Oracle Academy Use Only

Regular Expressions Functions and Conditions: Syntax

```
REGEXP_LIKE   (source_char, pattern [,match_option]
```

```
REGEXP_INSTR  (source_char, pattern [, position
               [, occurrence [, return_option
               [, match_option [, subexpr]]]]])
```

```
REGEXP_SUBSTR (source_char, pattern [, position
               [, occurrence [, match_option
               [, subexpr]]]])
```

```
REGEXP_REPLACE(source_char, pattern [,replacestr
                [, position [, occurrence
                [, match_option]]]])
```

```
REGEXP_COUNT (source_char, pattern [, position

               [, occurrence [, match_option]]])
```

9

The syntax for the regular expressions functions and conditions is as follows:

- `source_char`: A character expression that serves as the search value
- `pattern`: A regular expression, a text literal
- `occurrence`: A positive integer indicating which occurrence of pattern in `source_char` Oracle Server should search for. The default is 1.
- `position`: A positive integer indicating the character of `source_char` where Oracle Server should begin the search. The default is 1.
- `return_option`:
  - 0: Returns the position of the first character of the occurrence (default)
  - 1: Returns the position of the character following the occurrence
- `Replacestr`: Character string replacing pattern
- `match_parameter`:
  - " `c` ": Uses case-sensitive matching (default)
  - " `i` ": Uses non-case-sensitive matching
  - " `n` ": Allows match-any-character operator
  - " `m` ": Treats source string as multiple lines
- `subexpr`: Fragment of pattern enclosed in parentheses. You learn more about subexpressions later in this appendix.

Oracle Internal & Oracle Academy Use Only

Performing a Basic Search by Using the REGEXP_LIKE Condition

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

10

`REGEXP_LIKE` is similar to the `LIKE` condition, except that `REGEXP_LIKE` performs regular-expression matching instead of the simple pattern matching performed by `LIKE`. This condition evaluates strings by using characters as defined by the input character set.

Example of `REGEXP_LIKE`

In this query, against the `EMPLOYEES` table, all employees with first names containing either Steven or Stephen are displayed. In the expression used `'^Ste(v|ph)en$ '`:

- `^` indicates the beginning of the expression
- `$` indicates the end of the expression
- `|` indicates either/or

Oracle Internal & Oracle Academy Use Only

Replacing Patterns by Using the REGEXP_REPLACE Function

```
REGEXP_REPLACE(source_char, pattern [,replacestr
[, position [, occurrence [, match_option]]]])
```

```
SELECT last_name, REGEXP_REPLACE(phone_number, '\.','-')
    AS phone
FROM employees;
```

Original

	LAST_NAME	PHONE
1	OConnell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

Partial results

	LAST_NAME	PHONE
1	OConnell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555

11

Using the REGEXP_REPLACE function, you reformat the phone number to replace the period (.) delimiter with a dash (–) delimiter. Here is an explanation of each of the elements used in the regular expression example:

- phone_number is the source column.
- '\.' is the search pattern.
  - Use single quotation marks (' ') to search for the literal character period (.).
  - Use a backslash (\) to search for a character that is normally treated as a metacharacter.
- '–' is the replace string.

Oracle Internal & Oracle Academy Use Only

Finding Patterns by Using the REGEXP_INSTR Function

```
REGEXP_INSTR  (source_char, pattern [, position [, occurrence [, return_option [,
    match_option]]]])
```

```
SELECT street_address,
REGEXP_INSTR(street_address,'[[:alpha:]]') AS
    First_Alpha_Position
FROM  locations;
```

	STREET_ADDRESS	FIRST_ALPHA_POSITION
1	1297 Via Cola di Rie	6
2	93091 Calle della Testa	7
3	2017 Shinjuku-ku	6
4	9450 Kamiya-cho	6

12

In this example, the `REGEXP_INSTR` function is used to search the street address to find the location of the first alphabetic character, regardless of whether it is in uppercase or lowercase. Note that `[:<class>:]` implies a character class and matches any character from within that class; `[:alpha:]` matches with any alphabetic character. The partial results are displayed.

In the expression used in the query `'[[:alpha:]]'`:

- `[` starts the expression
- `[:alpha:]` indicates alphabetic character class
- `]` ends the expression

Note: The POSIX character class operator enables you to search for an expression within a character list that is a member of a specific POSIX character class. You can use this operator to search for specific formatting, such as uppercase characters, or you can search for special characters such as digits or punctuation characters. The full set of POSIX character classes is supported. Use the syntax `[:class:]`, where `class` is the name of the POSIX character class to search for. The following regular expression searches for one or more consecutive uppercase characters : `[[:upper:]]+`.

Oracle Internal & Oracle Academy Use Only

Extracting Substrings by Using the REGEXP_SUBSTR Function

```
REGEXP_SUBSTR (source_char, pattern [, position
               [, occurrence [, match_option]]])
```

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ') AS Road
FROM locations;
```

	ROAD
1	Via
2	Calle
3	(null)
4	(null)
5	labberwocky

13

In this example, the road names are extracted from the LOCATIONS table. To do this, the contents in the STREET_ADDRESS column that are after the first space are returned by using the REGEXP_SUBSTR function. In the expression used in the query ' [^]+ ':

- [starts the expression
- ^ indicates NOT
- ' ' indicates space
-] ends the expression
- + indicates 1 or more

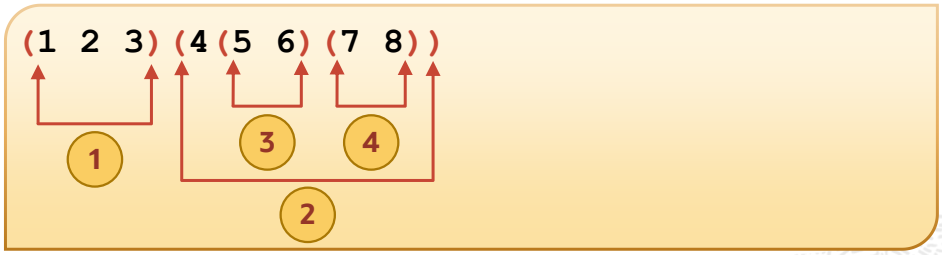
Oracle Internal & Oracle Academy Use Only

Subexpressions

Examine this expression:

```
(1 2 3)(4(5 6)(7 8))
```

The subexpressions are:



14

Oracle Database provides regular expression support parameters to access a subexpression. In the example in the slide, a string of digits is shown. The parentheses identify the subexpressions within the string of digits. Reading from left to right, and from outer parentheses to the inner parentheses, the subexpressions in the string of digits are:

1. 123
2. 45678
3. 56
4. 78

You can search for any of those subexpressions with the `REGEXP_INSTR` and `REGEXP_SUBSTR` functions.

Oracle Internal & Oracle Academy Use Only

Using Subexpressions with Regular Expression Support

```
SELECT
  REGEXP_INSTR
1 ('0123456789',       -- source char or search value
2 '(123)(4(56)(78))', -- regular expression patterns
3 1,                   -- position to start searching
4 1,                   -- occurrence
5 0,                   -- return option
6 'i',                 -- match option (case insensitive)
7 1)               -- subexpression on which to search
    "Position"
FROM dual;
```

	Position
1	2

15

`REGEXP_INSTR` and `REGEXP_SUBSTR` have an optional `SUBEXPR` parameter that lets you target a particular substring of the regular expression being evaluated.

In the example in the slide, you may want to search for the first subexpression pattern in your list of subexpressions. The example shown identifies several parameters for the `REGEXP_INSTR` function.

1. The string you are searching is identified.
2. The subexpressions are identified. The first subexpression is `123`. The second subexpression is `45678`, the third is `56`, and the fourth is `78`.
3. The third parameter identifies from which position to start searching.
4. The fourth parameter identifies the occurrence of the pattern you want to find. `1` means find the first occurrence.
5. The fifth parameter is the return option. This is the position of the first character of the occurrence. (If you specify `1`, the position of the character following the occurrence is returned.)
6. The sixth parameter identifies whether your search should be case-sensitive or not.
7. The last parameter specifies which subexpression you want to find. In the example shown, you are searching for the first subexpression, which is `123`.

Oracle Internal & Oracle Academy Use Only

Why Access the nth Subexpression?

- A more realistic use: DNA sequencing
- You may need to find a specific subpattern that identifies a protein needed for immunity in mouse DNA.

```
SELECT
    REGEXP_INSTR('ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgc
    ccccttaccctgcagggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatcctt
    gggctgcagagagaggagaatttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttga
    gttcatagcatttcttgagttttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtaggg
    ttgccctcaagaggctcttgggtctgatggccacatcctggaattgttttcaagttgatggtcacagccctgagg
    catgtaggggcgtggggatgcgctctgctctgctctcctctcctgaacccctgaaccctctggctaccccagagc
    acttagagccag',
                        '(gtc(tcac)(aaag))',
                     1, 1, 0, 'i',
                     1) "Position"
FROM dual;
```

	Position
1	195

16

In life sciences, you may need to extract the offsets of subexpression matches from a DNA sequence for further processing. For example, you may need to find a specific protein sequence, such as the begin offset for the DNA sequence preceded by `gtc` and followed by `tcac`, followed by `aaag`. To accomplish this goal, you can use the `REGEXP_INSTR` function, which returns the position where a match is found.

In the example in the slide, the position of the first subexpression `(gtc)` is returned. `gtc` appears starting in position 195 of the DNA string.

If you modify the example in the slide to search for the second subexpression `(tcac)`, the query results in the following output. `tcac` appears starting in position 198 of the DNA string.

	Position
1	198

If you modify the example in the slide to search for the third subexpression `(aaag)`, the query results in the following output. `aaag` appears starting in position 202 of the DNA string.

	Position
1	202

Oracle Internal & Oracle Academy Use Only

REGEXP_SUBSTR: Example

```
SELECT
  REGEXP_SUBSTR
  ('acgctgcactgca', -- source char or search value
   'acg(.*)gca',    -- regular expression pattern
   1,               -- position to start searching
   1,               -- occurrence
   'i',             -- match option (case insensitive)
   1)               -- sub-expression
 "Value"
FROM dual;
```

	Value
1	ctgcact

17

In the example in the slide:

1. `acgctgcactgca` is the source to be searched.
2. `acg(.*)gca` is the pattern to be searched. Find `acg` followed by `gca` with potential characters between the `acg` and the `gca`.
3. Start searching at the first character of the source.
4. Search for the first occurrence of the pattern.
5. Use non-case-sensitive matching on the source.
6. Use a nonnegative integer value that identifies the *n*th subexpression to be targeted. This is the subexpression parameter. In this example, `1` indicates the first subexpression. You can use a value from 0–9. A zero means that no subexpression is targeted. The default value for this parameter is 0.

Oracle Internal & Oracle Academy Use Only

Using the REGEXP_COUNT Function

```
REGEXP_COUNT (source_char, pattern [, position
                [, occurrence [, match_option]]])
```

```
SELECT REGEXP_COUNT(
    'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttaccctgcag
    ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagaggagaat
    ttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttcacc
    ctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtctgatggcca
    catcctggaattgttttcaagttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctgctctgctctcct
    ctcctgaacccctgaaccctctggctaccccagagcacttagagccag',
    'gtc') AS Count
FROM dual;
```

	COUNT
1	4

18

The REGEXP_COUNT function evaluates strings by using characters as defined by the input character set. It returns an integer indicating the number of occurrences of the pattern. If no match is found, the function returns 0.

In the example in the slide, the number of occurrences for a DNA substring is determined by using the REGEXP_COUNT function.

The following example shows that the number of times the pattern 123 occurs in the string 123123123123 is three times. The search starts from the second position of the string.

```
SELECT REGEXP_COUNT
   ('123123123123', -- source char or search value
   '123',           -- regular expression pattern
    2,              -- position where the search should start
   'i')             -- match option (case insensitive)
   As Count
FROM dual;
```

	COUNT
1	3

Oracle Internal & Oracle Academy Use Only

Regular Expressions and Check Constraints: Examples

```
ALTER TABLE emp8
  ADD CONSTRAINT email_addr
  CHECK(REGEXP_LIKE(email,'@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES
  (500,'Christian','Patel','ChrisP2creme.com',
   1234567890,'12-Jan-2004','HR_REP',2000,null,102,40);
```

```
Error starting at line 1 in command:
INSERT INTO emp8 VALUES
  (500,'Christian','Patel',
  'ChrisP2creme.com', 1234567890,
   '12-Jan-2004', 'HR_REP', 2000, null, 102, 40)
Error report:
SQL Error: ORA-02290: check constraint (TEACH_B.EMAIL_ADDR) violated
02290. 00000 -  "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check

*Action:   do not insert values that violate the constraint.
```

19

Regular expressions can also be used in `CHECK` constraints. In this example, a `CHECK` constraint is added on the `EMAIL` column of the `EMPLOYEES` table. This ensures that only strings containing an "`@`" symbol are accepted. The constraint is tested. The `CHECK` constraint is violated because the email address does not contain the required symbol. The `NOVALIDATE` clause ensures that existing data is not checked.

For the example in the slide, the `emp8` table is created by using the following code:

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

Note: The slide example is executed by using the Execute Statement option in SQL Developer. The output format differs if you use the Run Script option.

Oracle Internal & Oracle Academy Use Only

Quiz

With the use of regular expressions in SQL and PL/SQL, you can:

a. Avoid intensive string processing of SQL result sets by middle-tier applications

b. Avoid data validation logic on the client

c. Enforce constraints on the server

20

Answer: a, b, c

Oracle Internal & Oracle Academy Use Only

Summary

In this appendix, you should have learned how to use regular expressions to search for, match, and replace strings.

21

In this appendix, you have learned to use the regular expression support features. Regular expression support is available in both SQL and PL/SQL.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only