

# **Oracle Database 11g: SQL Fundamentals I**

**Volume II • Student Guide**

D49996GC11

Edition 1.1

April 2009

D59981

**ORACLE®**

## Authors

Puja Singh  
Brian Pottle

## Technical Contributors and Reviewers

Claire Bennett  
Tom Best  
Purjanti Chang  
Ken Cooper  
László Czinkóczki  
Burt Demchick  
Mark Fleming  
Gerlinde Frenzen  
Nancy Greenberg  
Chaitanya Koratamaddi  
Wendy Lo  
Timothy Mcglue  
Alan Paulson  
Bryan Roberts  
Abhishek Singh  
Lori Tritz  
Michael Versaci  
Lex van der Werff

## Editors

Raj Kumar  
Amitha Narayan  
Vijayalakshmi Narasimhan

## Graphic Designer

Satish Bettgowda

## Publishers

Sujatha Nagendra  
Syed Ali

Copyright © 2009, Oracle. All rights reserved.

## Disclaimer

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

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

## Restricted Rights Notice

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

### U.S. GOVERNMENT RIGHTS

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

## Trademark Notice

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

Oracle Internal & Oracle Academy  
Use Only

# Contents

## Preface

### I Introduction

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

Summary 1-39  
Practice I: Overview 1-40

## **1 Retrieving Data Using the SQL `SELECT` Statement**

Objectives 1-2  
Lesson Agenda 1-3  
Capabilities of SQL `SELECT` Statements 1-4  
Basic `SELECT` Statement 1-5  
Selecting All Columns 1-6  
Selecting Specific Columns 1-7  
Writing SQL Statements 1-8  
Column Heading Defaults 1-9  
Lesson Agenda 1-10  
Arithmetic Expressions 1-11  
Using Arithmetic Operators 1-12  
Operator Precedence 1-13  
Defining a Null Value 1-14  
Null Values in Arithmetic Expressions 1-15  
Lesson Agenda 1-16  
Defining a Column Alias 1-17  
Using Column Aliases 1-18  
Lesson Agenda 1-19  
Concatenation Operator 1-20  
Literal Character Strings 1-21  
Using Literal Character Strings 1-22  
Alternative Quote (`q`) Operator 1-23  
Duplicate Rows 1-24  
Lesson Agenda 1-25  
Displaying the Table Structure 1-26  
Using the `DESCRIBE` Command 1-27  
Quiz 1-28  
Summary 1-29  
Practice 1: Overview 1-30

## **2 Restricting and Sorting Data**

Objectives 2-2  
Lesson Agenda 2-3  
Limiting Rows Using a Selection 2-4  
Limiting the Rows That Are Selected 2-5  
Using the `WHERE` Clause 2-6

Character Strings and Dates 2-7  
Comparison Operators 2-8  
Using Comparison Operators 2-9  
Range Conditions Using the `BETWEEN` Operator 2-10  
Membership Condition Using the `IN` Operator 2-11  
Pattern Matching Using the `LIKE` Operator 2-12  
Combining Wildcard Characters 2-13  
Using the `NULL` Conditions 2-14  
Defining Conditions Using the Logical Operators 2-15  
Using the `AND` Operator 2-16  
Using the `OR` Operator 2-17  
Using the `NOT` Operator 2-18  
Lesson Agenda 2-19  
Rules of Precedence 2-20  
Lesson Agenda 2-22  
Using the `ORDER BY` Clause 2-23  
Sorting 2-24  
Lesson Agenda 2-26  
Substitution Variables 2-27  
Using the Single-Ampersand Substitution Variable 2-29  
Character and Date Values with Substitution Variables 2-31  
Specifying Column Names, Expressions, and Text 2-32  
Using the Double-Ampersand Substitution Variable 2-33  
Lesson Agenda 2-34  
Using the `DEFINE` Command 2-35  
Using the `VERIFY` Command 2-36  
Quiz 2-37  
Summary 2-38  
Practice 2: Overview 2-39

### **3 Using Single-Row Functions to Customize Output**

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

Character-Manipulation Functions 3-13  
Using the Character-Manipulation Functions 3-14  
Lesson Agenda 3-15  
Number Functions 3-16  
Using the `ROUND` Function 3-17  
Using the `TRUNC` Function 3-18  
Using the `MOD` Function 3-19  
Lesson Agenda 3-20  
Working with Dates 3-21  
`RR` Date Format 3-22  
Using the `SYSDATE` Function 3-24  
Arithmetic with Dates 3-25  
Using Arithmetic Operators with Dates 3-26  
Lesson Agenda 3-27  
Date-Manipulation Functions 3-28  
Using Date Functions 3-29  
Using `ROUND` and `TRUNC` Functions with Dates 3-30  
Quiz 3-31  
Summary 3-32  
Practice 3: Overview 3-33

#### **4 Using Conversion Functions and Conditional Expressions**

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

Using the NVL2 Function 4-30  
Using the NULLIF Function 4-31  
Using the COALESCE Function 4-32  
Lesson Agenda 4-35  
Conditional Expressions 4-36  
CASE Expression 4-37  
Using the CASE Expression 4-38  
DECODE Function 4-39  
Using the DECODE Function 4-40  
Quiz 4-42  
Summary 4-43  
Practice 4: Overview 4-44

## **5 Reporting Aggregated Data Using the Group Functions**

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

## **6 Displaying Data from Multiple Tables**

Objectives 6-2

Lesson Agenda 6-3

Obtaining Data from Multiple Tables 6-4

Types of Joins 6-5

Joining Tables Using SQL:1999 Syntax 6-6

Qualifying Ambiguous Column Names 6-7

Lesson Agenda 6-8

Creating Natural Joins 6-9

Retrieving Records with Natural Joins 6-10

Creating Joins with the `USING` Clause 6-11

Joining Column Names 6-12

Retrieving Records with the `USING` Clause 6-13

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

Creating Joins with the `ON` Clause 6-15

Retrieving Records with the `ON` Clause 6-16

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

Applying Additional Conditions to a Join 6-18

Lesson Agenda 6-19

Joining a Table to Itself 6-20

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

Lesson Agenda 6-22

Nonequijoins 6-23

Retrieving Records with Nonequijoins 6-24

Lesson Agenda 6-25

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

`INNER` Versus `OUTER` Joins 6-27

`LEFT OUTER JOIN` 6-28

`RIGHT OUTER JOIN` 6-29

`FULL OUTER JOIN` 6-30

Lesson Agenda 6-31

Cartesian Products 6-32

Generating a Cartesian Product 6-33

Creating Cross Joins 6-34

Quiz 6-35

Summary 6-36

Practice 6: Overview 6-37



## **7 Using Subqueries to Solve Queries**

Objectives 7-2

Lesson Agenda 7-3

Using a Subquery to Solve a Problem 7-4

Subquery Syntax 7-5

Using a Subquery 7-6

Guidelines for Using Subqueries 7-7

Types of Subqueries 7-8

Lesson Agenda 7-9

Single-Row Subqueries 7-10

Executing Single-Row Subqueries 7-11

Using Group Functions in a Subquery 7-12

The `HAVING` Clause with Subqueries 7-13

What Is Wrong with This Statement? 7-14

No Rows Returned by the Inner Query 7-15

Lesson Agenda 7-16

Multiple-Row Subqueries 7-17

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

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

Lesson Agenda 7-20

Null Values in a Subquery 7-21

Quiz 7-23

Summary 7-24

Practice 7: Overview 7-25

## **8 Using the Set Operators**

Objectives 8-2

Lesson Agenda 8-3

Set Operators 8-4

Set Operator Guidelines 8-5

The Oracle Server and Set Operators 8-6

Lesson Agenda 8-7

Tables Used in This Lesson 8-8

Lesson Agenda 8-12

`UNION` Operator 8-13

Using the `UNION` Operator 8-14

`UNION ALL` Operator 8-16

Using the `UNION ALL` Operator 8-17

Lesson Agenda 8-18

`INTERSECT` Operator 8-19

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

## **9 Manipulating Data**

Objectives 9-2  
Lesson Agenda 9-3  
Data Manipulation Language 9-4  
Adding a New Row to a Table 9-5  
`INSERT` Statement Syntax 9-6  
Inserting New Rows 9-7  
Inserting Rows with Null Values 9-8  
Inserting Special Values 9-9  
Inserting Specific Date and Time Values 9-10  
Creating a Script 9-11  
Copying Rows from Another Table 9-12  
Lesson Agenda 9-13  
Changing Data in a Table 9-14  
`UPDATE` Statement Syntax 9-15  
Updating Rows in a Table 9-16  
Updating Two Columns with a Subquery 9-17  
Updating Rows Based on Another Table 9-18  
Lesson Agenda 9-19  
Removing a Row from a Table 9-20  
`DELETE` Statement 9-21  
Deleting Rows from a Table 9-22  
Deleting Rows Based on Another Table 9-23  
`TRUNCATE` Statement 9-24  
Lesson Agenda 9-25  
Database Transactions 9-26  
Database Transactions: Start and End 9-27

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

## **10 Using DDL Statements to Create and Manage Tables**

Objectives	10-2
Lesson Agenda	10-3
Database Objects	10-4
Naming Rules	10-5
Lesson Agenda	10-6
CREATE TABLE Statement	10-7
Referencing Another User's Tables	10-8
DEFAULT Option	10-9
Creating Tables	10-10
Lesson Agenda	10-11
Data Types	10-12
Datetime Data Types	10-14
Lesson Agenda	10-15
Including Constraints	10-16
Constraint Guidelines	10-17
Defining Constraints	10-18
NOT NULL Constraint	10-20
UNIQUE Constraint	10-21
PRIMARY KEY Constraint	10-23

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

## **11 Creating Other Schema Objects**

Objectives 11-2  
Lesson Agenda 11-3  
Database Objects 11-4  
What Is a View? 11-5  
Advantages of Views 11-6  
Simple Views and Complex Views 11-7  
Creating a View 11-8  
Retrieving Data from a View 11-11  
Modifying a View 11-12  
Creating a Complex View 11-13  
Rules for Performing DML Operations on a View 11-14  
Using the WITH CHECK OPTION Clause 11-17  
Denying DML Operations 11-18  
Removing a View 11-20  
Practice 11: Overview of Part 1 11-21  
Lesson Agenda 11-22  
Sequences 11-23  
CREATE SEQUENCE Statement: Syntax 11-25  
Creating a Sequence 11-26  
NEXTVAL and CURRVAL Pseudocolumns 11-27  
Using a Sequence 11-29  
Caching Sequence Values 11-30  
Modifying a Sequence 11-31

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

## **Appendix A: Practice Solutions**

## **Appendix B: Table Descriptions**

## **Appendix C: Oracle Join Syntax**

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

Summary C-22  
Practice C: Overview C-23

## **Appendix D: Using SQL\*Plus**

Objectives D-2  
SQL and SQL\*Plus Interaction D-3  
SQL Statements Versus SQL\*Plus Commands D-4  
Overview of SQL\*Plus D-5  
Logging In to SQL\*Plus D-6  
Displaying Table Structure D-8  
SQL\*Plus Editing Commands D-10  
Using LIST, n, and APPEND D-12  
Using the CHANGE Command D-13  
SQL\*Plus File Commands D-14  
Using the SAVE, START, and EDIT Commands D-15  
SERVEROUTPUT Command D-17  
Using the SQL\*Plus SPOOL Command D-18  
Using the AUTOTRACE Command D-19  
Summary D-20

## **Appendix E: Using SQL Developer**

Objectives E-2  
What Is Oracle SQL Developer? E-3  
Specifications of SQL Developer E-4  
Installing SQL Developer E-5  
SQL Developer 1.2 Interface E-6  
Creating a Database Connection E-7  
Browsing Database Objects E-10  
Creating a Schema Object E-11  
Creating a New Table: Example E-12  
Using the SQL Worksheet E-13  
Executing SQL Statements E-16  
Saving SQL Scripts E-17  
Executing Saved Script Files: Method 1 E-18  
Executing Saved Script Files: Method 2 E-19  
Executing SQL Statements E-20  
Formatting the SQL Code E-21  
Using Snippets E-22  
Using Snippets: Example E-23  
Using SQL\*Plus E-24

Debugging Procedures and Functions	E-25
Database Reporting	E-26
Creating a User-Defined Report	E-27
Search Engines and External Tools	E-28
Setting Preferences	E-29
Specifications of SQL Developer 1.5.3	E-30
Installing SQL Developer 1.5.3	E-31
SQL Developer 1.5.3 Interface	E-32
Summary	E-34

## **Index**

## **Additional Practices**

## **Additional Practices: Solutions**

Oracle Internal & Oracle Academy  
Use Only





# 9

## Manipulating Data

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy  
Use Only

## Objectives

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

- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Control transactions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Objective

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

## Lesson Agenda

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Data Manipulation Language

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction*.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decreasing the savings account, increasing the checking account, and recording the transaction in the transaction journal. The Oracle server must guarantee that all the three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

### Note

- Most of the DML statements in this lesson assume that no constraints on the table are violated. Constraints are discussed later in this course.
- In SQL Developer, click the Run Script icon or press [F5] to run the DML statements. The feedback messages will be shown on the Script Output tabbed page.

## Adding a New Row to a Table

70 Public Relations	100	1700
---------------------	-----	------

New  
row

## 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

Insert new row  
into the  
DEPARTMENTS table.

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

9	70 Public Relations	100	1700
---	---------------------	-----	------

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Adding a New Row to a Table

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

## INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement:

```
INSERT INTO  table [(column [, column...])]  
VALUES      (value [, value...]);
```

- With this syntax, only one row is inserted at a time.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INSERT Statement Syntax

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

In the syntax:

<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to populate
<i>value</i>	is the corresponding value for the column

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

## Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
INSERT INTO departments(department_id,  
                        department_name, manager_id, location_id)  
VALUES (70, 'Public Relations', 100, 1700);
```

1 rows inserted

- Enclose character and date values within single quotation marks.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Inserting New Rows

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

DESCRIBE departments

Name	Null	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

For clarity, use the column list in the `INSERT` clause.

Enclose character and date values within single quotation marks; however, it is not recommended that you enclose numeric values within single quotation marks.

## Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO departments (department_id,  
                        department_name)  
VALUES (30, 'Purchasing');
```

1 rows inserted

- Explicit method: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO departments  
VALUES (100, 'Finance', NULL, NULL);
```

1 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Inserting Rows with Null Values

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the NULL keyword in the VALUES list; specify the empty string ('') in the VALUES list for character strings and dates.

Be sure that you can use null values in the targeted column by verifying the Null status with the DESCRIBE command.

The Oracle server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input are checked in the following order:

- Mandatory value missing for a NOT NULL column
- Duplicate value violating any unique or primary key constraint
- Any value violating a CHECK constraint
- Referential integrity maintained for foreign key constraint
- Data type mismatches or values too wide to fit in column

**Note:** Use of the column list is recommended as it makes the INSERT statement more readable and reliable, or less prone to mistakes.



# Inserting Special Values

The SYSDATE function records the current date and time.

```
INSERT INTO employees (employee_id,
                        first_name, last_name,
                        email, phone_number,
                        hire_date, job_id, salary,
                        commission_pct, manager_id,
                        department_id)
VALUES (113,
        'Louis', 'Popp',
        'LPOPP', '515.124.4567',
        SYSDATE, 'AC_ACCOUNT', 6900,
        NULL, 205, 110);
```

1 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Inserting Special Values

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

The slide example records information for employee Popp in the EMPLOYEES table. It supplies the current date and time in the HIRE\_DATE column. It uses the SYSDATE function that returns the current date and time of the database server. You may also use the CURRENT\_DATE function to get the current date in the session time zone. You can also use the USER function when inserting rows in a table. The USER function records the current username.

### Confirming Additions to the Table

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

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
1	113	Popp	AC_ACCOUNT	11-JUN-07	(null)

## Inserting Specific Date and Time Values

- Add a new employee.

```
INSERT INTO employees
VALUES
    (114,
     'Den', 'Raphealy',
     'DRAPHEAL', '515.127.4561',
     TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
     'SA_REP', 11000, 0.2, 100, 60);
```

1 rows inserted

- Verify your addition.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	SA_REP	11000	0.2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Inserting Specific Date and Time Values

The DD-MON-RR format is generally used to insert a date value. With the RR format, the system provides the correct century automatically.

You may also supply the date value in the DD-MON-YYYY format. This is recommended because it clearly specifies the century and does not depend on the internal RR format logic of specifying the correct century.

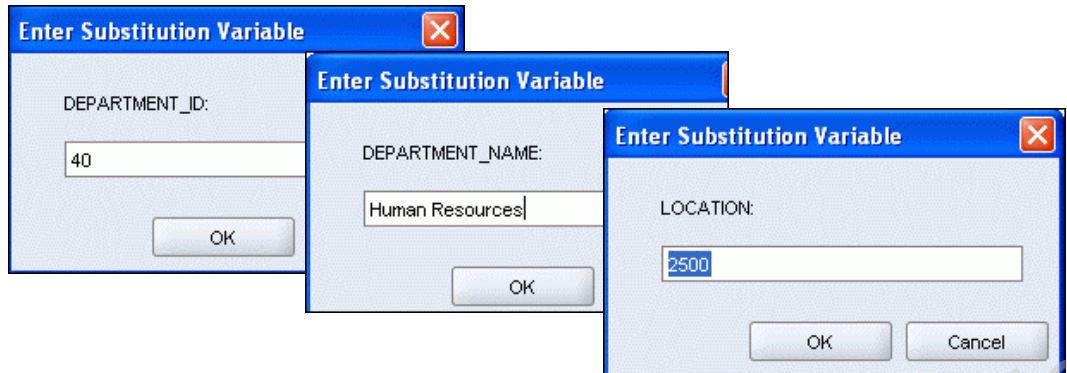
If a date must be entered in a format other than the default format (for example, with another century or a specific time), you must use the TO\_DATE function.

The example in the slide records information for employee Raphealy in the EMPLOYEES table. It sets the HIRE\_DATE column to be February 3, 1999.

## Creating a Script

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
INSERT INTO departments
      (department_id, department_name, location_id)
VALUES (&department_id, '&department_name', &location);
```



The image shows three overlapping 'Enter Substitution Variable' dialog boxes. The first dialog prompts for DEPARTMENT\_ID with the value 40. The second dialog prompts for DEPARTMENT\_NAME with the value Human Resources. The third dialog prompts for LOCATION with the value 2500.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Creating a Script

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

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

## Copying Rows from Another Table

- Write your INSERT statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

4 rows inserted

- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.
- Inserts all the rows returned by the subquery in the table, sales\_reps.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Copying Rows from Another Table

You can use the INSERT statement to add rows to a table where the values are derived from existing tables. In the slide example, for the INSERT INTO statement to work, you must have already created the sales\_reps table using the CREATE TABLE statement. CREATE TABLE is discussed in the next lesson titled “Using DDL Statements to Create and Manage Tables.”

In place of the VALUES clause, you use a subquery.

#### Syntax

```
INSERT INTO table [ column (, column) ] subquery;
```

In the syntax:

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

The number of columns and their data types in the column list of the INSERT clause must match the number of values and their data types in the subquery. Zero or more rows are added depending on the number of rows returned by the subquery. To create a copy of the rows of a table, use SELECT \* in the subquery:

```
INSERT INTO copy_emp
SELECT *
FROM employees;
```

## Lesson Agenda

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Changing Data in a Table

## EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

Update rows in the EMPLOYEES table:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Changing Data in a Table

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

## UPDATE Statement Syntax

- Modify existing values in a table with the UPDATE statement:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- Update more than one row at a time (if required).

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### UPDATE Statement Syntax

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

In the syntax:

<i>table</i>	is the name of the table
<i>column</i>	is the name of the column in the table to populate
<i>value</i>	is the corresponding value or subquery for the column
<i>condition</i>	identifies the rows to be updated and is composed of column names, expressions, constants, subqueries, and comparison operators

Confirm the update operation by querying the table to display the updated rows.

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

**Note:** In general, use the primary key column in the WHERE clause to identify a single row for update. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

## Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the WHERE clause:

```
UPDATE employees
SET    department_id = 50
WHERE  employee_id = 113;
```

1 rows updated

- Values for all the rows in the table are modified if you omit the WHERE clause:

```
UPDATE    copy_emp
SET       department_id = 110;
```

22 rows updated

- Specify SET *column\_name* = NULL to update a column value to NULL.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Updating Rows in a Table

The UPDATE statement modifies the values of a specific row or rows if the WHERE clause is specified. The example in the slide shows the transfer of employee 113 (Popp) to department 50.

If you omit the WHERE clause, values for all the rows in the table are modified. Examine the updated rows in the COPY\_EMP table.

```
SELECT last_name, department_id
FROM    copy_emp;
```

	LAST_NAME	DEPARTMENT_ID
1	King	110
2	Kochhar	110

...

For example, an employee who was a SA\_REP has now changed his job to an IT\_PROG. Therefore, his JOB\_ID needs to be updated and the commission field needs to be set to NULL.

```
UPDATE employees
SET job_id = 'IT_PROG', commission_pct = NULL
WHERE employee_id = 114;
```

**Note:** The COPY\_EMP table has the same data as the EMPLOYEES table.



## Updating Two Columns with a Subquery

Update employee 113's job and salary to match those of employee 205.

```
UPDATE employees
SET   job_id = (SELECT job_id
                FROM   employees
                WHERE  employee_id = 205),
      salary = (SELECT salary
                FROM   employees
                WHERE  employee_id = 205)
WHERE employee_id = 113;
```

1 rows updated

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Updating Two Columns with a Subquery

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

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

The example in the slide can also be written as follows:

```
UPDATE employees
SET (job_id, salary) = (SELECT job_id, salary
                       FROM   employees
                       WHERE  employee_id = 205)
WHERE employee_id = 113;
```

## Updating Rows Based on Another Table

Use the subqueries in the `UPDATE` statements to update row values in a table based on values from another table:

```
UPDATE copy_emp
SET    department_id = (SELECT department_id
                        FROM employees
                        WHERE employee_id = 100)
WHERE  job_id        = (SELECT job_id
                        FROM employees
                        WHERE employee_id = 200);
```

1 rows updated

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Updating Rows Based on Another Table

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

## Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Removing a Row from a Table

## DEPARTMENTS

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

Delete a row from the DEPARTMENTS table:

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Removing a Row from a Table

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

# DELETE Statement

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## DELETE Statement Syntax

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

In the syntax:

*table* is the name of the table  
*condition* identifies the rows to be deleted, and is composed of column names, expressions, constants, subqueries, and comparison operators

**Note:** If no rows are deleted, the message “0 rows deleted” is returned (in the Script Output tab in SQL Developer)

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

## Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
DELETE FROM departments
WHERE department_name = 'Finance';
```

1 rows deleted

- All rows in the table are deleted if you omit the WHERE clause:

```
DELETE FROM copy_emp;
```

22 rows deleted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Deleting Rows from a Table

You can delete specific rows by specifying the WHERE clause in the DELETE statement. The first example in the slide deletes the Accounting department from the DEPARTMENTS table. You can confirm the delete operation by displaying the deleted rows using the SELECT statement.

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
```

0 rows selected

However, if you omit the WHERE clause, all rows in the table are deleted. The second example in the slide deletes all rows from the COPY\_EMP table, because no WHERE clause was specified.

#### Example:

Remove rows identified in the WHERE clause.

```
DELETE FROM employees WHERE employee_id = 114;
```

1 rows deleted

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

2 rows deleted

## Deleting Rows Based on Another Table

Use the subqueries in the `DELETE` statements to remove rows from a table based on values from another table:

```
DELETE FROM employees
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name
           LIKE '%Public%');
1 rows deleted
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Deleting Rows Based on Another Table

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

The subquery searches the `DEPARTMENTS` table to find the department number based on the department name containing the string `Public`. The subquery then feeds the department number to the main query, which deletes rows of data from the `EMPLOYEES` table based on this department number.

## TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TRUNCATE Statement

A more efficient method of emptying a table is by using the TRUNCATE statement.

You can use the TRUNCATE statement to quickly remove all rows from a table or cluster. Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information. Rollback information is covered later in this lesson.
- Truncating a table does not fire the delete triggers of the table.

If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement. Disabling constraints is covered in a subsequent lesson.



## Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- **Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT**
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Database Transactions

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that constitute one consistent change to the data. For example, a transfer of funds between two accounts should include the debit in one account and the credit to another account of the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

### Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only one DCL statement

## Database Transactions: Start and End

- Begin when the first DML SQL statement is executed.
- End with one of the following events:
  - A COMMIT or ROLLBACK statement is issued.
  - A DDL or DCL statement executes (automatic commit).
  - The user exits SQL Developer or SQL\*Plus.
  - The system crashes.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Database Transaction: Start and End

When does a database transaction start and end?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A COMMIT or ROLLBACK statement is issued.
- A DDL statement, such as CREATE, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL\*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

## **Advantages of COMMIT and ROLLBACK Statements**

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically-related operations

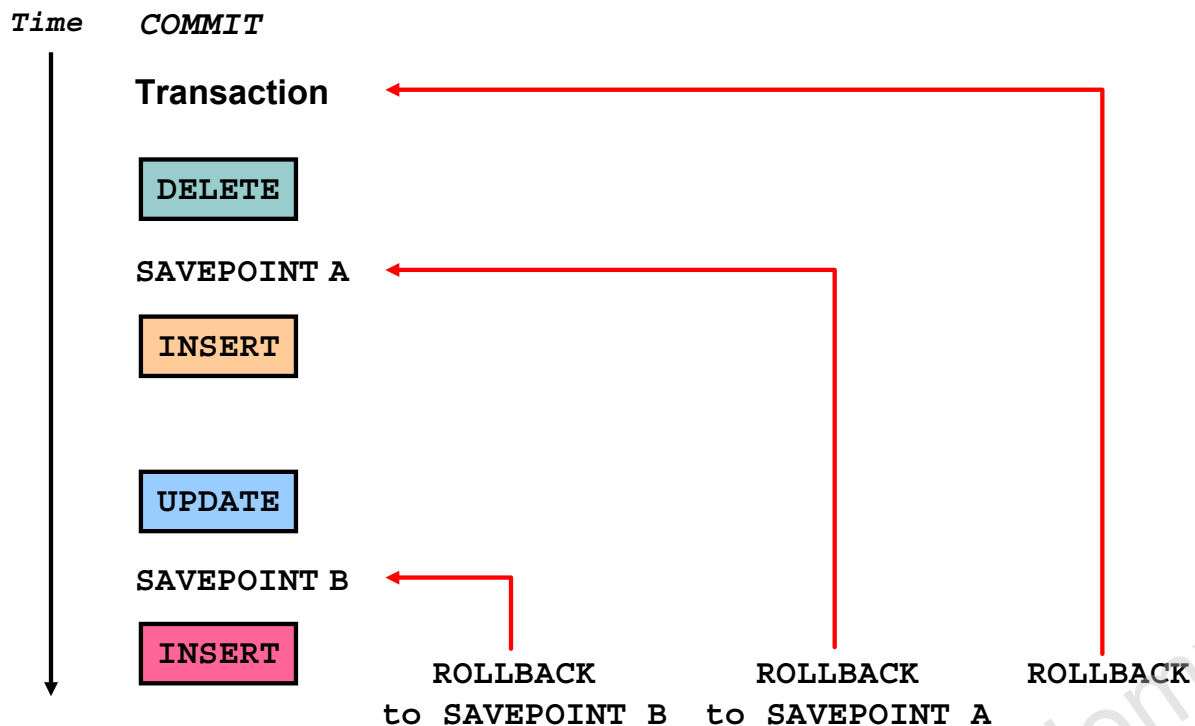
ORACLE

Copyright © 2009, Oracle. All rights reserved.

### **Advantages of COMMIT and ROLLBACK Statements**

With the COMMIT and ROLLBACK statements, you have control over making changes to the data permanent.

# Explicit Transaction Control Statements



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Explicit Transaction Control Statements

You can control the logic of transactions by using the **COMMIT**, **SAVEPOINT**, and **ROLLBACK** statements.

Statement	Description
<b>COMMIT</b>	Ends the current transaction by making all pending data changes permanent
<b>SAVEPOINT</b> <i>name</i>	Marks a savepoint within the current transaction
<b>ROLLBACK</b>	<b>ROLLBACK</b> ends the current transaction by discarding all pending data changes.
<b>ROLLBACK TO</b> <i>SAVEPOINT name</i>	<b>ROLLBACK TO SAVEPOINT</b> rolls back the current transaction to the specified savepoint, thereby discarding any changes and/or savepoints that were created after the savepoint to which you are rolling back. If you omit the <b>TO SAVEPOINT</b> clause, the <b>ROLLBACK</b> statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

**Note:** You cannot **COMMIT** to a **SAVEPOINT**. **SAVEPOINT** is not ANSI-standard SQL.

## Rolling Back Changes to a Marker

- Create a marker in the current transaction by using the `SAVEPOINT` statement.
- Roll back to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

```
UPDATE...  
SAVEPOINT update_done;  
SAVEPOINT update_done succeeded.  
INSERT...  
ROLLBACK TO update_done;  
ROLLBACK TO succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Rolling Back Changes to a Marker

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

Note, if you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

## Implicit Transaction Processing

- An automatic commit occurs in the following circumstances:
  - A DDL statement is issued
  - A DCL statement is issued
  - Normal exit from SQL Developer or SQL\*Plus, without explicitly issuing `COMMIT` or `ROLLBACK` statements
- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL\*Plus or a system failure.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Implicit Transaction Processing

Status	Circumstances
Automatic commit	DDL statement or DCL statement issued SQL Developer or SQL*Plus exited normally, without explicitly issuing <code>COMMIT</code> or <code>ROLLBACK</code> commands
Automatic rollback	Abnormal termination of SQL Developer or SQL*Plus or system failure

**Note:** In SQL\*Plus, the `AUTOCOMMIT` command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the `COMMIT` statement can still be issued explicitly. Also, the `COMMIT` statement is issued when a DDL statement is issued or when you exit SQL\*Plus. The `SET AUTOCOMMIT ON/OFF` command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled. To enable Autocommit, perform the following:

- In the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.
- On the right pane, check the Autocommit in SQL Worksheet option. Click OK.

## **Implicit Transaction Processing (continued)**

### **System Failures**

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to the state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL\*Plus, a normal exit is accomplished by entering the `EXIT` command at the prompt. Closing the window is interpreted as an abnormal exit.

Oracle Internal & Oracle Academy  
Use Only



## State of the Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the `SELECT` statement.
- Other users *cannot* view the results of the DML statements issued by the current user.
- The affected rows are *locked*; other users cannot change the data in the affected rows.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### State of the Data Before COMMIT or ROLLBACK

Every data change made during the transaction is temporary until the transaction is committed.

The state of the data before COMMIT or ROLLBACK statements are issued can be described as follows:

- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit.
- The affected rows are locked; other users cannot change the data in the affected rows.

## State of the Data After COMMIT

- Data changes are saved in the database.
- The previous state of the data is overwritten.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### State of the Data After COMMIT

Make all pending changes permanent by using the COMMIT statement. Here is what happens after a COMMIT statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All users can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
- All savepoints are erased.

## Committing Data

- Make the changes:

```
DELETE FROM employees
WHERE employee_id = 99999;
1 rows deleted

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

- Commit the changes:

```
COMMIT;
COMMIT succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Committing Data

In the example in the slide, a row is deleted from the EMPLOYEES table and a new row is inserted into the DEPARTMENTS table. The changes are saved by issuing the COMMIT statement.

#### Example:

Remove departments 290 and 300 in the DEPARTMENTS table and update a row in the EMPLOYEES table. Save the data change.

```
DELETE FROM departments
WHERE department_id IN (290, 300);
```

```
UPDATE employees
SET department_id = 80
WHERE employee_id = 206;
```

```
COMMIT;
```

## State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;  
ROLLBACK ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement, which results in the following:

- Data changes are undone.
- The previous state of the data is restored.
- Locks on the affected rows are released.

## State of the Data After ROLLBACK: Example

```
DELETE FROM test;  
25,000 rows deleted.  
  
ROLLBACK;  
Rollback complete.  
  
DELETE FROM test WHERE id = 100;  
1 row deleted.  
  
SELECT * FROM test WHERE id = 100;  
No rows selected.  
  
COMMIT;  
Commit complete.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### State of the Data After ROLLBACK: Example

While attempting to remove a record from the TEST table, you may accidentally empty the table. However, you can correct the mistake, reissue a proper statement, and make the data change permanent.

## Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Statement-Level Rollback

A part of a transaction can be discarded through an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the transaction are not discarded. They can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

## Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- **Read consistency**
- FOR UPDATE clause in a SELECT statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with the changes made by another user.
- Read consistency ensures that, on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers
  - Writers wait for writers

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Read Consistency

Database users access the database in two ways:

- Read operations (SELECT statement)
- Write operations (INSERT, UPDATE, DELETE statements)

You need read consistency so that the following occur:

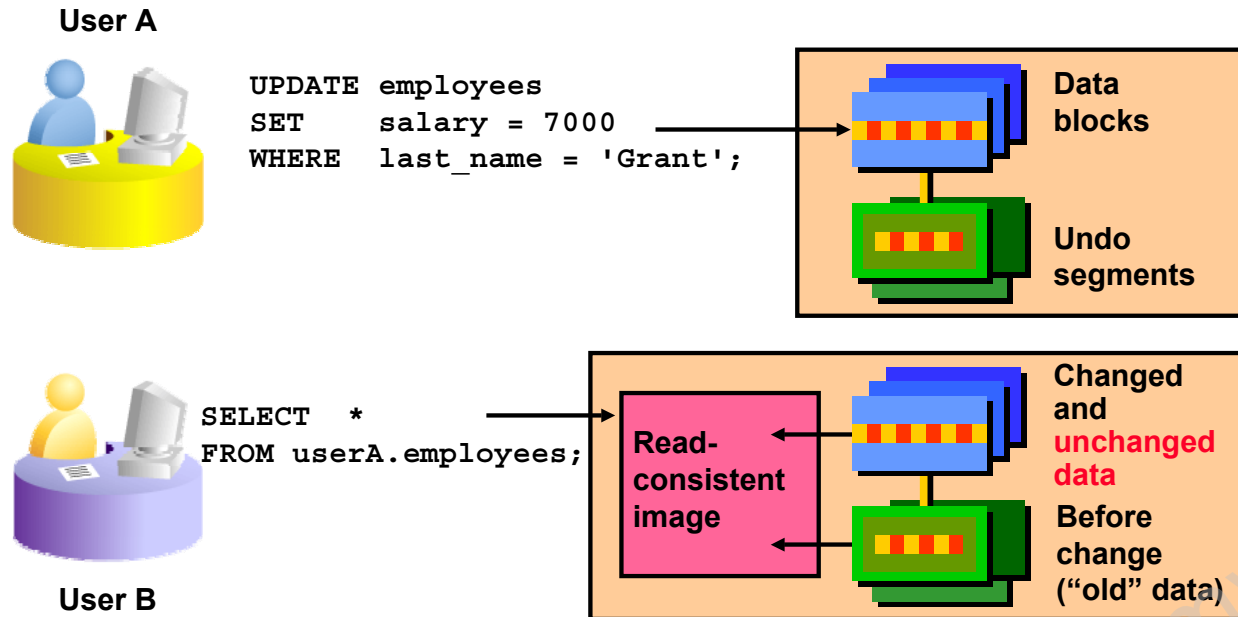
- The database reader and writer are ensured a consistent view of the data.
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent manner.
- Changes made by one writer do not disrupt or conflict with the changes being made by another writer.

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

**Note:** The same user can login as different sessions. Each session maintains read consistency in the manner described above, even if they are the same users.



# Implementing Read Consistency



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Implementing Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in the undo segments. The read-consistent image is constructed from the committed data in the table and the old data that is being changed and is not yet committed from the undo segment.

When an insert, update, or delete operation is made on the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

Before the changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a *SELECT* statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:

- The original, older version of the data in the undo segment is written back to the table.
- All users see the database as it existed before the transaction began.

## Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- **FOR UPDATE clause in a SELECT statement**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the EMPLOYEES table where job\_id is SA\_REP.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

- Lock is released only when you issue a ROLLBACK or a COMMIT.
- If the SELECT statement attempts to lock a row that is locked by another user, then the database waits until the row is available, and then returns the results of the SELECT statement.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOR UPDATE Clause in a SELECT Statement

When you issue a SELECT statement against the database to query some records, no locks are placed on the selected rows. In general, this is required because the number of records locked at any given time is (by default) kept to the absolute minimum: only those records that have been changed but not yet committed are locked. Even then, others will be able to read those records as they appeared before the change (the “before image” of the data). There are times, however, when you may want to lock a set of records even before you change them in your program. Oracle offers the FOR UPDATE clause of the SELECT statement to perform this locking.

When you issue a SELECT . . . FOR UPDATE statement, the relational database management system (RDBMS) automatically obtains exclusive row-level locks on all the rows identified by the SELECT statement, thereby holding the records “for your changes only.” No one else will be able to change any of these records until you perform a ROLLBACK or a COMMIT.

You can append the optional keyword NOWAIT to the FOR UPDATE clause to tell the Oracle server not to wait if the table has been locked by another user. In this case, control will be returned immediately to your program or to your SQL Developer environment so that you can perform other work, or simply wait for a period of time before trying again. Without the NOWAIT clause, your process will block until the table is available, when the locks are released by the other user through the issue of a COMMIT or a ROLLBACK command.

## FOR UPDATE Clause: Examples

- You can use the `FOR UPDATE` clause in a `SELECT` statement against multiple tables.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

- Rows from both the `EMPLOYEES` and `DEPARTMENTS` tables are locked.
- Use `FOR UPDATE OF column_name` to qualify the column you intend to change, then only the rows from that specific table are locked.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOR UPDATE Clause: Examples

In the example in the slide, the statement locks rows in the `EMPLOYEES` table with `JOB_ID` set to `ST_CLERK` and `LOCATION_ID` set to 1500, and locks rows in the `DEPARTMENTS` table with departments in `LOCATION_ID` set as 1500.

You can use the `FOR UPDATE OF column_name` to qualify the column that you intend to change. The `OF` list of the `FOR UPDATE` clause does not restrict you to changing only those columns of the selected rows. Locks are still placed on all rows; if you simply state `FOR UPDATE` in the query and do not include one or more columns after the `OF` keyword, the database will lock all identified rows across all the tables listed in the `FROM` clause.

The following statement locks only those rows in the `EMPLOYEES` table with `ST_CLERK` located in `LOCATION_ID` 1500. No rows are locked in the `DEPARTMENTS` table:

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE OF e.salary
ORDER BY e.employee_id;
```

### **FOR UPDATE Clause: Examples (continued)**

In the following example, the database is instructed to wait for five seconds for the row to become available, and then return control to you.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE WAIT 5
ORDER BY employee_id;
```

Oracle Internal & Oracle Academy  
Use Only

## Quiz

The following statements produce the same results:

```
DELETE FROM copy_emp;
```

```
TRUNCATE TABLE copy_emp;
```

1. True
2. False

ORACLE

Copyright © 2009, Oracle. All rights reserved.

**Answer: 2**

## Summary

In this lesson, you should have learned how to use the following statements:

Function	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
TRUNCATE	Removes all rows from a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes
FOR UPDATE clause in SELECT	Locks rows identified by the SELECT query

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to manipulate data in the Oracle database by using the INSERT, UPDATE, DELETE, and TRUNCATE statements, as well as how to control data changes by using the COMMIT, SAVEPOINT, and ROLLBACK statements. You also learned how to use the FOR UPDATE clause of the SELECT statement to lock rows for your changes only.

Remember that the Oracle server guarantees a consistent view of data at all times.

## Practice 9: Overview

This practice covers the following topics:

- Inserting rows into the tables
- Updating and deleting rows in the table
- Controlling transactions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Practice 9: Overview

In this practice, you add rows to the MY\_EMPLOYEE table, update and delete data from the table, and control your transactions. You run a script to create the MY\_EMPLOYEE table.



## Practice 9

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, you use the MY\_EMPLOYEE table before giving the statements to the HR department.

**Note:** For all the DML statements, use the Run Script icon (or press [F5]) to execute the query. This way you get to see the feedback messages on the Script Output tab page. For SELECT queries, continue to use the Execute Statement icon or press [F9] to get the formatted output on the Results tab page.

### Insert data into the MY\_EMPLOYEE table.

1. Run the statement in the lab\_09\_01.sql script to build the MY\_EMPLOYEE table used in this practice.
2. Describe the structure of the MY\_EMPLOYEE table to identify the column names.

DESCRIBE MY_EMPLOYEE		
Name	Null	Type
-----		
ID	NOT NULL	NUMBER(4)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
SALARY		NUMBER(9,2)






3. Create an INSERT statement to add *the first row* of data to the MY\_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause. *Do not enter all rows yet.*

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550






4. Populate the MY\_EMPLOYEE table with the second row of the sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

## Practice 9 (continued)

- Confirm your addition to the table.

	 ID	 LAST_NAME	 FIRST_NAME	 USERID	 SALARY
1	1	Patel	Ralph	rpatel	895
2	2	Dancs	Betty	bdancs	860






- Write an INSERT statement in a dynamic reusable script file to load the remaining rows into the MY\_EMPLOYEE table. The script should prompt for all the columns (ID, LAST\_NAME, FIRST\_NAME, USERID, and SALARY). Save this script to a lab\_09\_06.sql file.
- Populate the table with the next two rows of the sample data listed in step 3 by running the INSERT statement in the script that you created.
- Confirm your additions to the table.

	 ID	 LAST_NAME	 FIRST_NAME	 USERID	 SALARY
1	1	Patel	Ralph	rpatel	895
2	2	Dancs	Betty	bdancs	860
3	3	Biri	Ben	bbiri	1100
4	4	Newman	Chad	cnewman	750



- Make the data additions permanent.

## Update and delete data in the MY\_EMPLOYEE table.

- Change the last name of employee 3 to Drexler.
- Change the salary to \$1,000 for all employees who have a salary less than \$900.
- Verify your changes to the table.

	 ID	 LAST_NAME	 FIRST_NAME	 USERID	 SALARY
1	1	Patel	Ralph	rpatel	1000
2	2	Dancs	Betty	bdancs	1000
3	3	Drexler	Ben	bbiri	1100
4	4	Newman	Chad	cnewman	1000

- Delete Betty Dancs from the MY\_EMPLOYEE table.
- Confirm your changes to the table.

	 ID	 LAST_NAME	 FIRST_NAME	 USERID	 SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000

## Practice 9 (continued)

- Commit all pending changes.

### Control data transaction to the MY\_EMPLOYEE table.

- Populate the table with the last row of the sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.
- Confirm your addition to the table.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000
4	5	Ropeburn	Audrey	aropebur	1550

- Mark an intermediate point in the processing of the transaction.
- Delete all the rows from the MY\_EMPLOYEE table.
- Confirm that the table is empty.
- Discard the most recent DELETE operation without discarding the earlier INSERT operation.
- Confirm that the new row is still intact.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000
4	5	Ropeburn	Audrey	aropebur	1550

- Make the data addition permanent.

If you have the time, complete the following exercise:

- Modify the lab\_09\_06.sql script such that the USERID is generated automatically by concatenating the first letter of the first name and the first seven characters of the last name. The generated USERID must be in lowercase. Hence, the script should not prompt for the USERID. Save this script to a file named lab\_09\_24.sql.
- Run the script, lab\_09\_24.sql to insert the following record:

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

- Confirm that the new row was added with correct USERID.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230



# 10

## Using DDL Statements to Create and Manage Tables

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy  
Use Only

## Objectives

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

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

In this lesson, you are introduced to the data definition language (DDL) statements. You are taught the basics of how to create simple tables, alter them, and remove them. The data types available in DDL are shown and schema concepts are introduced. Constraints are discussed in this lesson. Exception messages that are generated from violating constraints during DML operations are shown and explained.

# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# 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 some queries
Synonym	Gives alternative name to an object

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Database Objects

The Oracle database can contain multiple data structures. Each structure should be outlined in the database design so that it can be created during the build stage of database development.

- **Table:** Stores data
- **View:** Subset of data from one or more tables
- **Sequence:** Generates numeric values
- **Index:** Improves the performance of some queries
- **Synonym:** Gives alternative name to an object

### Oracle Table Structures

- Tables can be created at any time, even when users are using the database.
- You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
- Table structure can be modified online.

**Note:** More database objects are available, but are not covered in this course.



# Naming Rules

Table names and column names:

- Must begin with a letter
- Must be 1–30 characters long
- Must contain only A–Z, a–z, 0–9, \_, \$, and #
- Must not duplicate the name of another object owned by the same user
- Must not be an Oracle server–reserved word

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Naming Rules

You name database tables and columns according to the standard rules for naming any Oracle database object:

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, \_ (underscore), \$, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must not be an Oracle server–reserved word.
  - You may also use quoted identifiers to represent the name of an object. A quoted identifier begins and ends with double quotation marks (“”). If you name a schema object using a quoted identifier, then you must use the double quotation marks whenever you refer to that object. Quoted identifiers can be reserved words, although this is not recommended.

## Naming Guidelines

Use descriptive names for tables and other database objects.

**Note:** Names are not case-sensitive. For example, EMPLOYEES is treated to be the same name as eMpLoYeeS or eMpLoYEEs. However, quoted identifiers are case-sensitive.

For more information, see the section on *Schema Object Names and Qualifiers* in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## CREATE TABLE Statement

- You must have:
  - CREATE TABLE privilege
  - A storage area

```
CREATE TABLE [schema.] table  
      (column datatype [DEFAULT expr] [, ...]);
```

- You specify:
  - Table name
  - Column name, column data type, and column size



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CREATE TABLE Statement

You create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the DDL statements that are a subset of the SQL statements used to create, modify, or remove Oracle database structures. These statements have an immediate effect on the database and they also record information in the data dictionary.

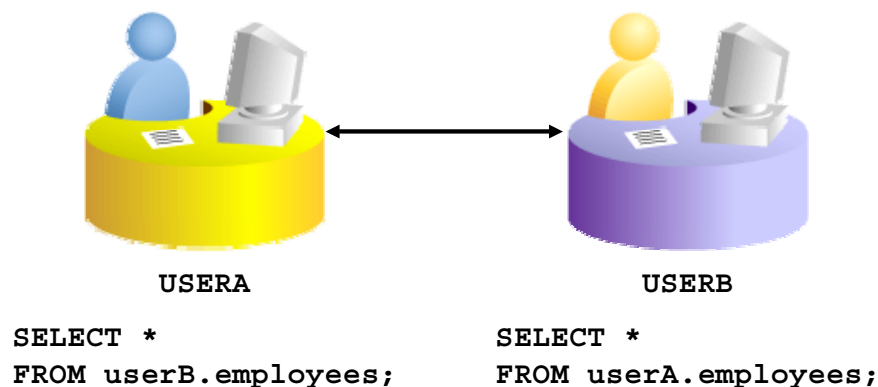
To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator (DBA) uses data control language (DCL) statements to grant privileges to users.

In the syntax:

<i>schema</i>	Is the same as the owner's name
<i>table</i>	Is the name of the table
DEFAULT <i>expr</i>	Specifies a default value if a value is omitted in the INSERT statement
<i>column</i>	Is the name of the column
<i>datatype</i>	Is the column's data type and length

## 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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Referencing Another User's Tables

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;
```

## DEFAULT Option

- Specify a default value for a column during an insert.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates  
  (id          NUMBER(8),  
   hire date DATE DEFAULT SYSDATE);
```

```
CREATE TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DEFAULT Option

When you define a table, you can specify that a column should be given a default value by using the DEFAULT option. This option prevents null values from entering the columns when a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudocolumn (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

Consider the following examples:

```
INSERT INTO hire_dates values(45, NULL);
```

The above statement will insert the null value rather than the default value.

```
INSERT INTO hire_dates(id) values(35);
```

The above statement will insert SYSDATE for the HIRE\_DATE column.

**Note:** In SQL Developer, click the Run Script icon or press [F5] to run the DDL statements. The feedback messages will be shown on the Script Output tabbed page.

# Creating Tables

- Create the table:

```
CREATE TABLE dept
  (deptno      NUMBER(2),
   dname       VARCHAR2(14),
   loc         VARCHAR2(13),
   create_date DATE DEFAULT SYSDATE);
```

CREATE TABLE succeeded.

- Confirm table creation:

```
DESCRIBE dept
```

DESCRIBE dept		
Name	Null	Type
-----		
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Creating Tables

The example in the slide creates the DEPT table with four columns: DEPTNO, DNAME, LOC, and CREATE\_DATE. The CREATE\_DATE column has a default value. If a value is not provided for an INSERT statement, the system date is automatically inserted.

To confirm that the table was created, run the DESCRIBE command.

Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- **Data types**
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Data Types

Data Type	Description
VARCHAR2 ( <i>size</i> )	Variable-length character data
CHAR ( <i>size</i> )	Fixed-length character data
NUMBER ( <i>p</i> , <i>s</i> )	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Data Types

When you identify a column for a table, you need to provide a data type for the column. There are several data types available:

Data Type	Description
VARCHAR2 ( <i>size</i> )	Variable-length character data (A maximum <i>size</i> must be specified: minimum <i>size</i> is 1; maximum <i>size</i> is 4,000.)
CHAR [( <i>size</i> )]	Fixed-length character data of length <i>size</i> bytes (Default and minimum <i>size</i> is 1; maximum <i>size</i> is 2,000.)
NUMBER [( <i>p</i> , <i>s</i> )]	Number having precision <i>p</i> and scale <i>s</i> (Precision is the total number of decimal digits and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38, and scale can range from -84 to 127.)
DATE	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)



## Data Types (continued)

Data Type	Description
RAW ( <i>size</i> )	Raw binary data of length <i>size</i> (A maximum <i>size</i> must be specified: maximum <i>size</i> is 2,000.)
LONG RAW	Raw binary data of variable length (up to 2 GB)
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

### Guidelines

- A LONG column is not copied when a table is created using a subquery.
- A LONG column cannot be included in a GROUP BY or an ORDER BY clause.
- Only one LONG column can be used per table.
- No constraints can be defined on a LONG column.
- You might want to use a CLOB column rather than a LONG column.

# Datetime Data Types

You can use several datetime data types:

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes, and seconds



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Datetime Data Types

Data Type	Description
TIMESTAMP	Enables storage of time as a date with fractional seconds. It stores the year, month, day, hour, minute, and the second value of the DATE data type as well as the fractional seconds value There are several variations of this data type such as WITH TIMEZONE, WITH LOCALTIMEZONE.
INTERVAL YEAR TO MONTH	Enables storage of time as an interval of years and months. Used to represent the difference between two datetime values in which the only significant portions are the year and month
INTERVAL DAY TO SECOND	Enables storage of time as an interval of days, hours, minutes, and seconds. Used to represent the precise difference between two datetime values

**Note:** These datetime data types are available with Oracle9i and later releases. The datetime data types are discussed in detail in the lesson titled “Managing Data in Different Time Zones” in the *Oracle Database 11g: SQL Fundamentals II* course.

Also, for more information about the datetime data types, see the topics *TIMESTAMP Datatype*, *INTERVAL YEAR TO MONTH Datatype*, and *INTERVAL DAY TO SECOND Datatype* in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Including Constraints

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Constraints

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables.
- Provide rules for Oracle tools, such as Oracle Developer.

### Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a referential integrity between the column and a column of the referenced table such that values in one table match values in another table.
CHECK	Specifies a condition that must be true

## Constraint Guidelines

- You can name a constraint, or the Oracle server generates a name by using the `SYS_Cn` format.
- Create a constraint at either of the following times:
  - At the same time as the creation of the table
  - After the creation of the table
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Constraint Guidelines

All constraints are stored in the data dictionary. Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules, except that the name cannot be the same as another object owned by the same user. If you do not name your constraint, the Oracle server generates a name with the format `SYS_Cn`, where *n* is an integer so that the constraint name is unique.

Constraints can be defined at the time of table creation or after the creation of the table. You can define a constraint at the column or table level. Functionally, a table-level constraint is the same as a column-level constraint.

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

# Defining Constraints

- Syntax:

```
CREATE TABLE [schema.]table
  (column datatype [DEFAULT expr]
   [column_constraint],
   ...
   [table_constraint] [,...]);
```

- Column-level constraint syntax:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint syntax:

```
column, ...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Defining Constraints

The slide gives the syntax for defining constraints when creating a table. You can create constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition and must refer to the column or columns on which the constraint pertains in a set of parentheses. It is mainly the syntax that differentiates the two; otherwise, functionally, a column-level constraint is the same as a table-level constraint.

NOT NULL constraints must be defined at the column level.

Constraints that apply to more than one column must be defined at the table level.

In the syntax:

<i>schema</i>	Is the same as the owner's name
<i>table</i>	Is the name of the table
DEFAULT <i>expr</i>	Specifies a default value to be used if a value is omitted in the INSERT statement
<i>column</i>	Is the name of the column
<i>datatype</i>	Is the column's data type and length
<i>column_constraint</i>	Is an integrity constraint as part of the column definition
<i>table_constraint</i>	Is an integrity constraint as part of the table definition

# Defining Constraints

- Example of a column-level constraint:

```
CREATE TABLE employees (  
  employee_id  NUMBER(6)  
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,  
  first_name   VARCHAR2(20),  
  ...);
```

1

- Example of a table-level constraint:

```
CREATE TABLE employees (  
  employee_id  NUMBER(6),  
  first_name   VARCHAR2(20),  
  ...  
  job_id       VARCHAR2(10) NOT NULL,  
  CONSTRAINT emp_emp_id_pk  
    PRIMARY KEY (EMPLOYEE_ID));
```

2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Defining Constraints (continued)

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also be temporarily disabled.

Both examples in the slide create a primary key constraint on the EMPLOYEE\_ID column of the EMPLOYEES table.

1. The first example uses the column-level syntax to define the constraint.
2. The second example uses the table-level syntax to define the constraint.

More details about the primary key constraint are provided later in this lesson.

## NOT NULL Constraint

Ensures that null values are not permitted for the column:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	COMMISSION_PCT
100	Steven	King	SKING	17-JUN-87	AD_PRES	(null)
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	(null)
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	(null)
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	(null)
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	(null)
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	(null)
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	(null)
141	Trenna	Rajs	TRAJS	17-OCT-95	ST_CLERK	(null)
142	Curtis	Davies	CDAVIES	29-JAN-97	ST_CLERK	(null)
143	Randall	Matos	RMATOS	15-MAR-98	ST_CLERK	(null)
144	Peter	Vargas	PVARGAS	09-JUL-98	ST_CLERK	(null)
149	Eleni	Zlotkey	EZLOTKEY	29-JAN-00	SA_MAN	0.2
174	Ellen	Abel	EABEL	11-MAY-96	SA_REP	0.3

...

↑  
**NOT NULL constraint  
(Primary Key enforces  
NOT NULL constraint.)**

↑  
**NOT NULL  
constraint**

↑  
**Absence of NOT NULL  
constraint (Any row can  
contain a null value for  
this column.)**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

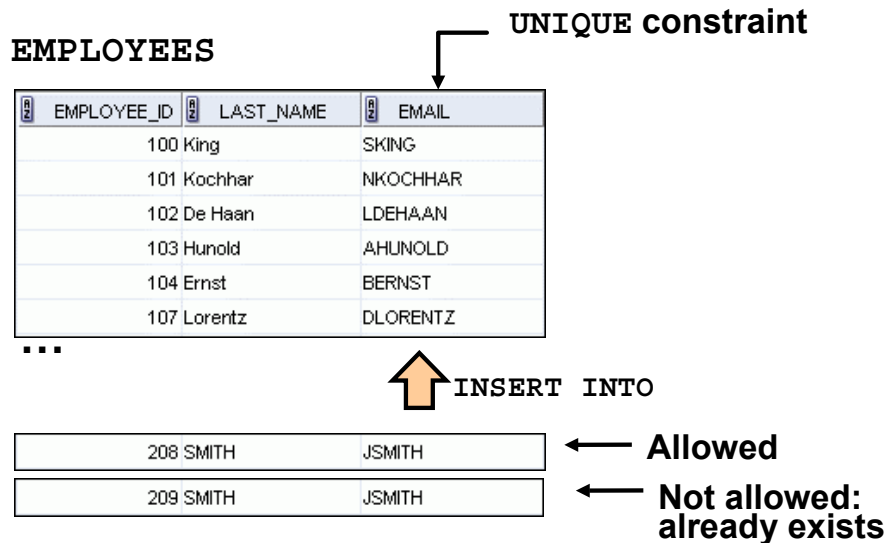
### NOT NULL Constraint

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level. In the EMPLOYEES table, the EMPLOYEE\_ID column inherits a NOT NULL constraint as it is defined as a primary key. Otherwise, the LAST\_NAME, EMAIL, HIRE\_DATE, and JOB\_ID columns have the NOT NULL constraint enforced on them.

**Note:** Primary key constraint is discussed in detail later in this lesson.



# UNIQUE Constraint



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## UNIQUE Constraint

A **UNIQUE** key integrity constraint requires that every value in a column or a set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or a set of columns. The column (or set of columns) included in the definition of the **UNIQUE** key constraint is called the *unique key*. If the **UNIQUE** constraint comprises more than one column, that group of columns is called a *composite unique key*.

**UNIQUE** constraints enable the input of nulls unless you also define **NOT NULL** constraints for the same columns. In fact, any number of rows can include nulls for columns without the **NOT NULL** constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite **UNIQUE** key) always satisfies a **UNIQUE** constraint.

**Note:** Because of the search mechanism for the **UNIQUE** constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite **UNIQUE** key constraint.

# UNIQUE Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(  
    employee_id    NUMBER(6),  
    last_name      VARCHAR2(25) NOT NULL,  
    email          VARCHAR2(25),  
    salary         NUMBER(8,2),  
    commission_pct NUMBER(2,2),  
    hire_date      DATE NOT NULL,  
    ...  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## UNIQUE Constraint (continued)

UNIQUE constraints can be defined at the column level or table level. You define the constraint at the table level when you want to create a composite unique key. A composite key is defined when there is not a single attribute that can uniquely identify a row. In that case, you can have a unique key that is composed of two or more columns, the combined value of which is always unique and can identify rows.

The example in the slide applies the UNIQUE constraint to the EMAIL column of the EMPLOYEES table. The name of the constraint is EMP\_EMAIL\_UK.

**Note:** The Oracle server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.

# PRIMARY KEY Constraint

DEPARTMENTS

PRIMARY KEY

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

Not allowed  
(null value)

↑ INSERT INTO

Public Accounting	124	2500
50 Finance	124	1500

Not allowed  
(50 already exists)

ORACLE

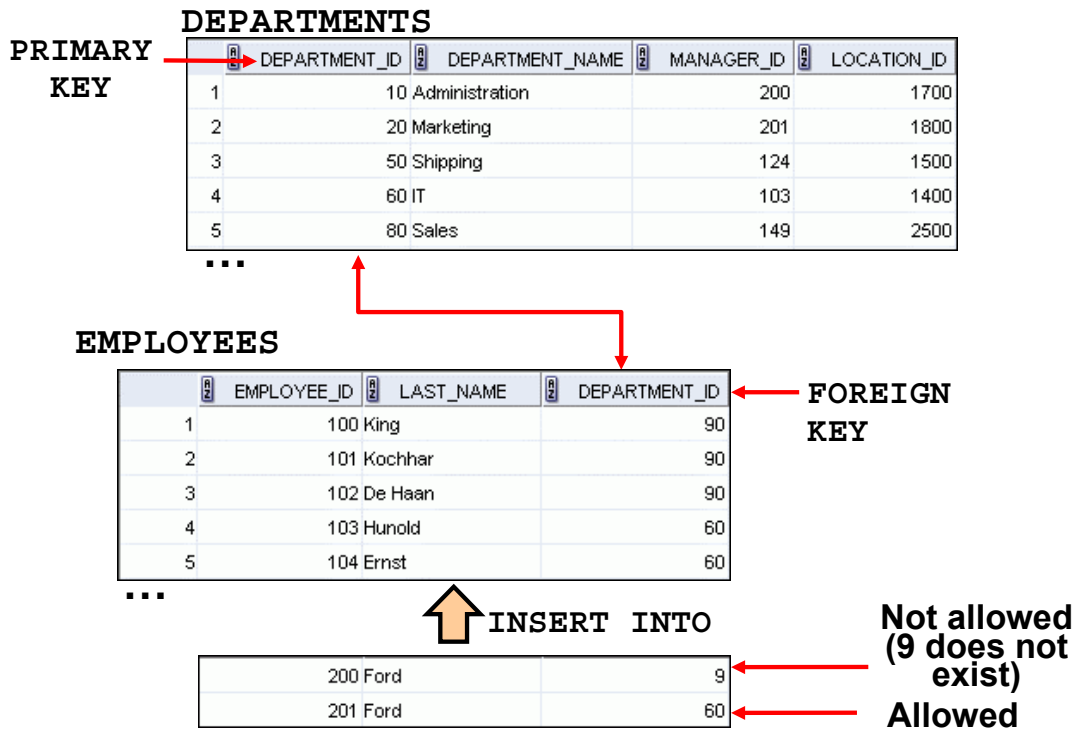
Copyright © 2009, Oracle. All rights reserved.

## PRIMARY KEY Constraint

A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for each table. The PRIMARY KEY constraint is a column or a set of columns that uniquely identifies each row in a table. This constraint enforces the uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.

**Note:** Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.

# FOREIGN KEY Constraint



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## FOREIGN KEY Constraint

The FOREIGN KEY (or referential integrity) constraint designates a column or a combination of columns as a foreign key and establishes a relationship with a primary key or a unique key in the same table or a different table.

In the example in the slide, DEPARTMENT\_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT\_ID column of the DEPARTMENTS table (the referenced or parent table).

### Guidelines

- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

# FOREIGN KEY Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(  
    employee_id      NUMBER(6),  
    last_name        VARCHAR2(25) NOT NULL,  
    email            VARCHAR2(25),  
    salary           NUMBER(8,2),  
    commission_pct   NUMBER(2,2),  
    hire_date        DATE NOT NULL,  
    ...  
    department_id    NUMBER(4),  
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)  
        REFERENCES departments(department_id),  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## FOREIGN KEY Constraint (continued)

FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.

The example in the slide defines a FOREIGN KEY constraint on the DEPARTMENT\_ID column of the EMPLOYEES table, using table-level syntax. The name of the constraint is EMP\_DEPT\_FK.

The foreign key can also be defined at the column level, provided that the constraint is based on a single column. The syntax differs in that the keywords FOREIGN KEY do not appear. For example:

```
CREATE TABLE employees  
(  
    ...  
    department_id NUMBER(4) CONSTRAINT emp_deptid_fk  
        REFERENCES departments(department_id),  
    ...  
)
```

## FOREIGN KEY Constraint: Keywords

- **FOREIGN KEY:** Defines the column in the child table at the table-constraint level
- **REFERENCES:** Identifies the table and column in the parent table
- **ON DELETE CASCADE:** Deletes the dependent rows in the child table when a row in the parent table is deleted
- **ON DELETE SET NULL:** Converts dependent foreign key values to null

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOREIGN KEY Constraint: Keywords

The foreign key is defined in the child table and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- **FOREIGN KEY** is used to define the column in the child table at the table-constraint level.
- **REFERENCES** identifies the table and the column in the parent table.
- **ON DELETE CASCADE** indicates that when a row in the parent table is deleted, the dependent rows in the child table are also deleted.
- **ON DELETE SET NULL** indicates that when a row in the parent table is deleted, the foreign key values are set to null.

The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the **ON DELETE CASCADE** or the **ON DELETE SET NULL** options, the row in the parent table cannot be deleted if it is referenced in the child table.

## CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - Calls to SYSDATE, UID, USER, and USERENV functions
  - Queries that refer to other values in other rows

```
..., salary NUMBER(2)
      CONSTRAINT emp_salary_min
      CHECK (salary > 0),...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CHECK Constraint

The CHECK constraint defines a condition that each row must satisfy. The condition can use the same constructs as the query conditions, with the following exceptions:

- References to the CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
- Calls to SYSDATE, UID, USER, and USERENV functions
- Queries that refer to other values in other rows

A single column can have multiple CHECK constraints that refer to the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

CHECK constraints can be defined at the column level or table level.

```
CREATE TABLE employees
(
  ...
  salary NUMBER(8,2) CONSTRAINT emp_salary_min
  CHECK (salary > 0),
  ...
)
```

## CREATE TABLE: Example

```
CREATE TABLE employees
( employee_id      NUMBER(6)
  CONSTRAINT emp_employee_id PRIMARY KEY
, first_name       VARCHAR2(20)
, last_name        VARCHAR2(25)
  CONSTRAINT emp_last_name_nn NOT NULL
, email            VARCHAR2(25)
  CONSTRAINT emp_email_nn    NOT NULL
  CONSTRAINT emp_email_uk    UNIQUE
, phone_number     VARCHAR2(20)
, hire_date        DATE
  CONSTRAINT emp_hire_date_nn NOT NULL
, job_id           VARCHAR2(10)
  CONSTRAINT emp_job_nn      NOT NULL
, salary           NUMBER(8,2)
  CONSTRAINT emp_salary_ck   CHECK (salary>0)
, commission_pct   NUMBER(2,2)
, manager_id       NUMBER(6)
  CONSTRAINT emp_manager_fk REFERENCES
    employees (employee_id)
, department_id    NUMBER(4)
  CONSTRAINT emp_dept_fk    REFERENCES
    departments (department_id));
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CREATE TABLE: Example

The example in the slide shows the statement that is used to create the EMPLOYEES table in the HR schema.



# Violating Constraints

```
UPDATE employees
SET   department_id = 55
WHERE department_id = 110;
```

Error starting at line 1 in command:

```
UPDATE employees
SET   department_id = 55
WHERE department_id = 110
```

Error report:

```
SQL Error: ORA-02291: integrity constraint (ORA16.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:      A foreign key value has no matching primary key value.
*Action:     Delete the foreign key or add a matching primary key.
```

Department 55 does not exist.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Violating Constraints

When you have constraints in place on columns, an error is returned if you try to violate the constraint rule. For example, if you try to update a record with a value that is tied to an integrity constraint, an error is returned.

In the example in the slide, department 55 does not exist in the parent table, DEPARTMENTS, and so you receive the “parent key not found” violation ORA-02291.

# Violating Constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
WHERE department_id = 60;
```

Error starting at line 1 in command: DELETE FROM departments WHERE department_id = 60 Error report: SQL Error: ORA-02292: integrity constraint (ORA16.EMP_DEPT_FK) violated - child record found 02292. 00000 - "integrity constraint (%s.%s) violated - child record found" *Cause: attempted to delete a parent key value that had a foreign dependency. *Action: delete dependencies first then parent or disable constraint.	
---	--

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Violating Constraints (continued)

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example in the slide tries to delete department 60 from the DEPARTMENTS table, but it results in an error because that department number is used as a foreign key in the EMPLOYEES table. If the parent record that you attempt to delete has child records, you receive the “child record found” violation ORA-02292.

The following statement works because there are no employees in department 70:

```
DELETE FROM departments
WHERE department_id = 70;
```

```
1 rows deleted
```

# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- **Creating a table using a subquery**
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Creating a Table Using a Subquery

- Create a table and insert rows by combining the `CREATE TABLE` statement and the `AS subquery` option.

```
CREATE TABLE table
      [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Table Using a Subquery

A second method for creating a table is to apply the `AS subquery` clause, which both creates the table and inserts rows returned from the subquery.

In the syntax:

*table* is the name of the table

*column* is the name of the column, default value, and integrity constraint

*subquery* is the `SELECT` statement that defines the set of rows to be inserted into the new table

#### Guidelines

- The table is created with the specified column names, and the rows retrieved by the `SELECT` statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery `SELECT` list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The column data type definitions and the `NOT NULL` constraint are passed to the new table. Note that only the explicit `NOT NULL` constraint will be inherited. The `PRIMARY KEY` column will not pass the `NOT NULL` feature to the new column. Any other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

## Creating a Table Using a Subquery

```
CREATE TABLE dept80
AS
  SELECT  employee_id, last_name,
          salary*12 ANNSAL,
          hire_date
  FROM    employees
  WHERE   department id = 80;
```

CREATE TABLE succeeded.

```
DESCRIBE dept80
```

Name	Null	Type
-----	-----	-----
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Table Using a Subquery (continued)

The example in the slide creates a table named DEPT80, which contains details of all the employees working in department 80. Notice that the data for the DEPT80 table comes from the EMPLOYEES table.

You can verify the existence of a database table and check the column definitions by using the DESCRIBE command.

However, be sure to provide a column alias when selecting an expression. The expression SALARY\*12 is given the alias ANNSAL. Without the alias, the following error is generated:

```
Error starting at line 1 in command:
CREATE TABLE    dept80
AS SELECT  employee_id, last_name,
salary*12,
hire_date FROM employees WHERE   department_id = 80
Error at Command Line:3 Column:6
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 - "must name this expression with a column alias"
*Cause:
*Action:
```

# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change table to read-only status

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ALTER TABLE Statement

After you create a table, you may need to change the table structure for any of the following reasons:

- You omitted a column.
- Your column definition or its name needs to be changed.
- You need to remove columns.
- You want to put the table into the read-only mode

You can do this by using the ALTER TABLE statement.

## Read-Only Tables

You can use the ALTER TABLE syntax to:

- Put a table into read-only mode, which prevents DDL or DML changes during table maintenance
- Put the table back into read/write mode

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Read-Only Tables

With Oracle Database 11g, you can specify READ ONLY to place a table in the read-only mode. When the table is in the READ-ONLY mode, you cannot issue any DML statements that affect the table or any SELECT . . . FOR UPDATE statements. You can issue DDL statements as long as they do not modify any data in the table. Operations on indexes associated with the table are allowed when the table is in the READ ONLY mode.

Specify READ/WRITE to return a read-only table to the read/write mode.

**Note:** You can drop a table that is in the READ ONLY mode. The DROP command is executed only in the data dictionary, so access to the table contents is not required. The space used by the table will not be reclaimed until the tablespace is made read/write again, and then the required changes can be made to the block segment headers, and so on.

For information about the ALTER TABLE statement, see the course titled *Oracle Database 10g SQL Fundamentals II*.



# Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Dropping a Table

- Moves a table to the recycle bin
- Removes the table and all its data entirely if the `PURGE` clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept80;
```

```
DROP TABLE dept80 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Dropping a Table

The `DROP TABLE` statement moves a table to the recycle bin or removes the table and all its data from the database entirely. Unless you specify the `PURGE` clause, the `DROP TABLE` statement does not result in space being released back to the tablespace for use by other objects, and the space continues to count towards the user's space quota. Dropping a table invalidates the dependent objects and removes object privileges on the table.

When you drop a table, the database loses all the data in the table and all the indexes associated with it.

#### Syntax

```
DROP TABLE table [PURGE]
```

In the syntax, *table* is the name of the table.

#### Guidelines

- All the data is deleted from the table.
- Any views and synonyms remain, but are invalid.
- Any pending transactions are committed.
- Only the creator of the table or a user with the `DROP ANY TABLE` privilege can remove a table.

**Note:** Use the `FLASHBACK TABLE` statement to restore a dropped table from the recycle bin. This is discussed in detail in the course titled *Oracle Database 11g: SQL Fundamentals II*.

## Quiz

You can use constraints to do the following:

1. Enforce rules on the data in a table whenever a row is inserted, updated, or deleted.
2. Prevent the deletion of a table.
3. Prevent the creation of a table.
4. Prevent the creation of data in a table.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

**Answer: 1, 2, 4**

## Summary

In this lesson, you should have learned how to use the `CREATE TABLE` statement to create a table and include constraints:

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Summary

In this lesson, you should have learned how to do the following:

### **CREATE TABLE**

- Use the `CREATE TABLE` statement to create a table and include constraints.
- Create a table based on another table by using a subquery.

### **DROP TABLE**

- Remove rows and a table structure.
- When executed, this statement cannot be rolled back.

## Practice 10: Overview

This practice covers the following topics:

- Creating new tables
- Creating a new table by using the `CREATE TABLE AS` syntax
- Verifying that tables exist
- Setting a table to read-only status
- Dropping tables

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Practice 10: Overview

Create new tables by using the `CREATE TABLE` statement. Confirm that the new table was added to the database. You also learn to set the status of a table as `READ ONLY` and then revert to `READ/WRITE`.

**Note:** For all the DDL and DML statements, click the Run Script icon (or press [F5]) to execute the query in SQL Developer. This way you get to see the feedback messages on the Script Output tab page. For `SELECT` queries, continue to click the Execute Statement icon or press [F9] to get the formatted output on the Results tab page.

## Practice 10

1. Create the DEPT table based on the following table instance chart. Save the statement in a script called `lab_10_01.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type	Primary key	
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

Name	Null	Type
-----	-----	-----
ID	NOT NULL	NUMBER(7)
NAME		VARCHAR2(25)

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.
3. Create the EMP table based on the following table instance chart. Save the statement in a script called `lab_10_03.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				DEPT
FK Column				ID
Data type	NUMBER	VARCHAR2	VARCHAR2	NUMBER
Length	7	25	25	7

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)

## Practice 10 (continued)

4. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPARTMENT\_ID columns. Name the columns in your new table ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPT\_ID, respectively.
5. Alter the EMPLOYEES2 table status to read-only.
6. Try to insert the following row in the EMPLOYEES2 table:

ID	FIRST_NAME	LAST_NAME	SALARY	DEPT_ID
34	Grant	Marcie	5678	10

You get the following error message:

```
Error starting at line 1 in command:
INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
Error at Command Line:1 Column:12
Error report:
SQL Error: ORA-12081: update operation not allowed on table "ORA16"."EMPLOYEES2"
12081. 00000 - "update operation not allowed on table \"%s\".\"%s\""
*Cause:      An attempt was made to update a read-only materialized view.
*Action:     No action required. Only Oracle is allowed to update a
              read-only materialized view.
```

7. Revert the EMPLOYEES2 table to the read/write status. Now, try to insert the same row again. You should get the following messages:

```
ALTER TABLE employees2 succeeded.
1 rows inserted
```

8. Drop the EMPLOYEES2 table.





# 11

## Creating Other Schema Objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy  
Use Only

## Objectives

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

- Create simple and complex views
- Retrieve data from views
- Create, maintain, and use sequences
- Create and maintain indexes
- Create private and public synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Objectives

In this lesson, you are introduced to the view, sequence, synonym, and index objects. You are taught the basics of creating and using views, sequences, and indexes.

# Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - Data manipulation language (DML) operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Database Objects

There are several other objects in a database in addition to tables. In this lesson, you learn about the views, sequences, indexes, and synonyms.

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.

# What Is a View?

**EMPLOYEES table**

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100 Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
2	101 Neena	Kochhar	NKOCHH...	515.123.4568	21-SEP-89	AD_VP	17000
3	102 Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
4	103 Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
5						IT_PROG	6000
6						IT_PROG	4200
7						ANALYST	5800
						MARKET	3500
						MARKET	3100
						MARKET	2600
						MARKET	2500
						MANAGER	10500
						SA_REP	11000
						SA_REP	8600
						SA_REP	7000
						AD_ASST	4400
						MK_MAN	13000
						MK_REP	6000
19	205 Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
20	206 William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACC...	8300

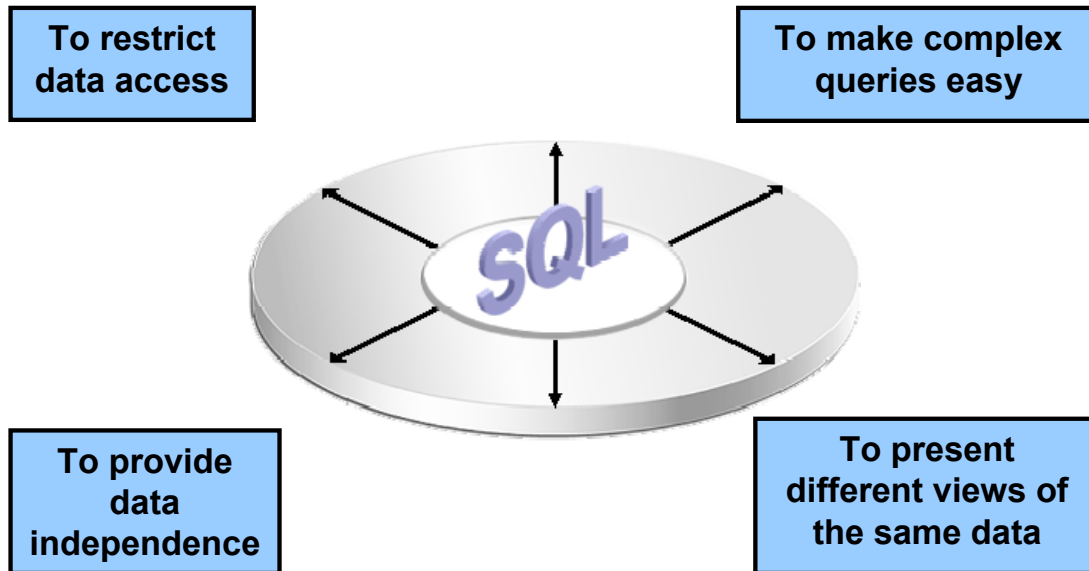
ORACLE

Copyright © 2009, Oracle. All rights reserved.

## What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table 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.

# Advantages of Views



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Advantages of Views

- Views restrict access to the data because it displays 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 section on “CREATE VIEW” in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

## 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Simple Views and Complex Views

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 is one that:
  - Derives data from only one table
  - Contains no functions or groups of data
  - Can perform DML operations through the view
- A complex view is one that:
  - Derives data from many tables
  - Contains functions or groups of data
  - Does not always allow DML operations through the view

## 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.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a View

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
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.)
<i>view</i>	Is the name of the view
<i>alias</i>	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.)
<i>subquery</i>	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
<i>constraint</i>	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.



## 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;
```

CREATE VIEW succeeded.

- Describe the structure of the view by using the *iSQL\*Plus* DESCRIBE command:

```
DESCRIBE empvu80
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a View (continued)

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\_Cn format.
- You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it, or regranting the object privileges previously granted on it.

## 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;
```

CREATE VIEW succeeded.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a View (continued)

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;
```

CREATE VIEW succeeded.

## Retrieving Data from a View

```
SELECT *  
FROM salvu50;
```

	<small>R 2</small>	ID_NUMBER	<small>R 2</small>	NAME	<small>R 2</small>	ANN_SALARY
1		124		Mourgos		69600
2		141		Rajs		42000
3		142		Davies		37200
4		143		Matos		31200
5		144		Vargas		30000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Retrieving Data from a View

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.

## 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;
CREATE OR REPLACE VIEW succeeded.
```

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Modifying a View

With the OR 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 regranting 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.

## 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
ON          (e.department_id = d.department_id)
GROUP BY d.department_name;
```

CREATE OR REPLACE VIEW succeeded.

ORACLE

Copyright © 2009, Oracle. All rights reserved.



### Creating a Complex View

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;
```

	NAME	MINSAL	MAXSAL	AVGSAL
1	Administration	4400	4400	4400
2	Accounting	8300	12000	10150
3	IT	4200	9000	6400
4	Executive	17000	24000	19333.3333333333333333...
5	Shipping	2500	5800	3500
6	Sales	8600	11000	10033.3333333333333333...
7	Marketing	6000	13000	9500

## 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 the following: 
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Rules for Performing DML Operations on a View

- You can perform DML operations on data through 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

## Rules for Performing DML 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
- Columns defined by expressions

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Rules for Performing DML Operations on a View (continued)

You can modify data through a view unless it contains any of the conditions mentioned in the previous slide or columns defined by expressions (for example, `SALARY * 12`).

## Rules for Performing DML Operations on a View

You cannot add data through 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 in the base tables that are not selected by the view

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Rules for Performing DML Operations on a View (continued)

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 section on “CREATE VIEW” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.



## Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay in the domain of the view by 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 ;
```

CREATE OR REPLACE VIEW succeeded.

- Any attempt to INSERT a row with a department\_id other than 20, or to UPDATE the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using the WITH CHECK OPTION Clause

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;
```

causes:

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

**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.

## Denying DML Operations

- You can ensure that no DML operations occur 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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the `WITH READ ONLY` option. The example in the next slide modifies the `EMPVU10` view to prevent any DML operations on the view.

# 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 ;
```

```
CREATE OR REPLACE VIEW succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Denying DML Operations (continued)

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
```

## Removing a View

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

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
```

```
DROP VIEW empvu80 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Removing a View

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. On the other hand, 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

## Practice 11: Overview of Part 1

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
- Removing views

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Practice 11: Overview of Part 1

Part 1 of this lesson's practice provides you with a variety of exercises in creating, using, and removing views. Complete questions 1–6 at the end of this lesson.

# Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - DML operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Sequences

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 some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

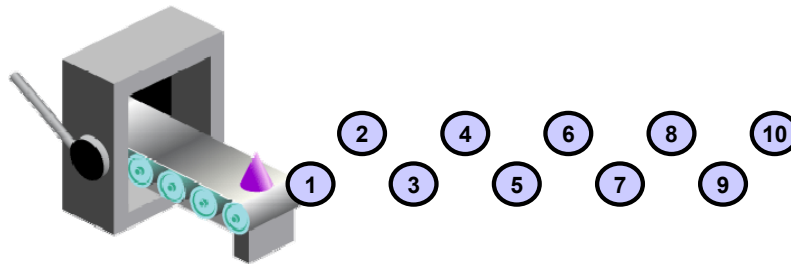
## Sequences

A sequence is a database object that creates integer values. You can create sequences and then use them to generate numbers.

# 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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Sequences (continued)

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.



## CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}] ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CREATE SEQUENCE Statement: Syntax

Automatically generate sequential numbers by using the CREATE SEQUENCE statement.

In the syntax:

<i>sequence</i>	Is the name of the sequence generator
INCREMENT BY <i>n</i>	Specifies the interval between sequence numbers, where <i>n</i> is an integer (If this clause is omitted, the sequence increments by 1.)
START WITH <i>n</i>	Specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)
MAXVALUE <i>n</i>	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 <i>n</i>	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.)

## 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  
    INCREMENT BY 10  
    START WITH 120  
    MAXVALUE 9999  
    NOCACHE  
    NOCYCLE;
```

```
CREATE SEQUENCE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Sequence (continued)

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 preallocates and keeps in memory (By default, the Oracle server caches 20 values.)

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 120, 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 section on “CREATE SEQUENCE” in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

**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.

## 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 contains a value.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### NEXTVAL and CURRVAL Pseudocolumns

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.

## **NEXTVAL and CURRVAL Pseudocolumns (continued)**

### **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

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 clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

For more information, see the sections on “Pseudocolumns” and “CREATE SEQUENCE” in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

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 rows inserted

- View the current value for the DEPT\_DEPTID\_SEQ sequence:

```
SELECT dept_deptid_seq.CURRVAL  
FROM dual;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Using a Sequence

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 follows.

You can view the current value of the sequence using the *sequence\_name*.CURRVAL, as shown in the second slide example. The output of the query is shown below:

	CURRVAL
1	120

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 EMPLOYEE\_SEQ has already been created to generate new employee numbers.

## 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Caching Sequence Values

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

Although sequence generators issue sequential numbers without gaps, this action occurs independent 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.

## 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;
```

```
ALTER SEQUENCE dept_deptid_seq succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Modifying a Sequence

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}];
```

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 11g, Release 1 (11.1)*.

## 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;  
DROP SEQUENCE dept_deptid_seq succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Guidelines for Modifying a Sequence

- 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 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;
```

- The error:

```
Error report:  
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
```



# Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - DML operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Indexes

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 some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

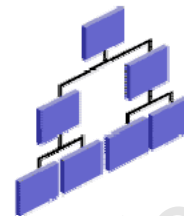
## Indexes

Indexes are database objects that you can create to improve the performance of some queries. Indexes can also be created automatically by the server when you create a primary key or a unique constraint.

# Indexes

An index:

- Is a schema object
- May 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 independent of the table that it indexes
- Is used and maintained automatically by the Oracle server



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Indexes (continued)

An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column, then 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 table that they index. This means that they can be created or dropped 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 11g, Release 1 (11.1)*.

## 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: Users can create nonunique indexes on columns to speed up access to the rows.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### How Are Indexes Created?

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.

## Creating an Index

- Create an index on one or more columns:

```
CREATE [UNIQUE] [BITMAP] 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);
```

```
CREATE INDEX succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating an Index

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) upon 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 rowids associated with a key value as a bitmap.

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

# Index Creation Guidelines

Create an index when:	
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a <code>WHERE</code> clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
Do not create an index when:	
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Index Creation Guidelines

### More Is Not Always Better

Having more indexes on a table does not produce faster queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes that you have associated with a table, the more effort the Oracle server must make to update all the indexes after a DML operation.

### When to Create an Index

Therefore, you should create indexes only if:

- The column contains a wide range of values
- The column contains a large number of null values
- One or more columns are frequently used together in a `WHERE` clause or join condition
- The table is large and most queries are expected to retrieve less than 2% to 4% of the rows

Remember that if you want to enforce uniqueness, you should define a unique constraint in the table definition. A unique index is then created automatically.

## 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;  
DROP INDEX emp_last_name_idx succeeded.
```

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Removing an Index

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

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.

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

# Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - DML operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.



# Synonyms

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 some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Synonyms

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.

## Creating a Synonym for an Object

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;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating a Synonym for an Object

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:

<b>PUBLIC</b>	Creates a synonym that is accessible to all users
<i>synonym</i>	Is the name of the synonym to be created
<i>object</i>	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.

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

## Creating and Removing Synonyms

- Create a shortened name for the DEPT\_SUM\_VU view:

```
CREATE SYNONYM d_sum  
FOR dept_sum_vu;
```

```
CREATE SYNONYM succeeded.
```

- Drop a synonym:

```
DROP SYNONYM d_sum;
```

```
DROP SYNONYM d_sum succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Creating and Removing Synonyms

### Creating a Synonym

The slide example creates a synonym for the DEPT\_SUM\_VU view 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
```

```
CREATE SYNONYM succeeded.
```

### 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 11g, Release 1 (11.1)*.

## Quiz

Indexes must be created manually and serve to speed up access to rows in a table.

1. True
2. False

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Answer: 2

**Note:** Indexes are designed to speed up query performance. However, not all indexes are created manually. The Oracle server automatically creates an index when you define a column in a table to have a PRIMARY KEY or a UNIQUE constraint.

## Summary

In this lesson, you should have learned how to:

- Create, use, and remove views
- Automatically generate sequence numbers by using a sequence generator
- Create indexes to improve speed of query retrieval
- Use synonyms to provide alternative names for objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Summary

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

## Practice 11: Overview of Part 2

This practice covers the following topics:

- Creating sequences
- Using sequences
- Creating nonunique indexes
- Creating synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Practice 11: Overview of Part 2

Part 2 of this lesson's practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym.

Complete questions 7–10 at the end of this lesson.

## Practice 11

### Part 1

1. The staff in the HR department wants to hide some of the data in the EMPLOYEES table. They want a view called EMPLOYEES\_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. They want the heading for the employee name to be EMPLOYEE.
2. Confirm that the view works. Display the contents of the EMPLOYEES\_VU view.

	EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60
5	104	Ernst	60

...

19	205	Higgins	110
20	206	Gietz	110

3. Using your EMPLOYEES\_VU view, write a query for the HR department to display all employee names and department numbers.

	EMPLOYEE	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60

...

19	Higgins	110
20	Gietz	110

## Practice 11 (continued)

- Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. You have been asked to label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.
- Display the structure and contents of the DEPT50 view.

Name	Null	Type
-----	-----	-----
EMPNO	NOT NULL	NUMBER(6)
EMPLOYEE	NOT NULL	VARCHAR2(25)
DEPTNO		NUMBER(4)

	EMPNO	EMPLOYEE	DEPTNO
1	124	Mourgos	50
2	141	Rajs	50
3	142	Davies	50
4	143	Matos	50
5	144	Vargas	50

- Test your view. Attempt to reassign Matos to department 80.



## Practice 11 (continued)

### Part 2

7. You need a sequence that can be used with the PRIMARY KEY column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT\_ID\_SEQ.
8. To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab\_11\_08.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.
9. Create a nonunique index on the NAME column in the DEPT table.
10. Create a synonym for your EMPLOYEES table. Call it EMP.

Oracle Internal & Oracle Academy  
Use Only

