
Oracle Database 12c: SQL Workshop I

Electronic Presentation

D80190GC11
Edition 1.1
July 2014

ORACLE®

Author

Dimpi Sarmah

**Technical Contributors
and Reviewers**

Nancy Greenberg

Swarnapriya Shridhar

Bryan Roberts

Laszlo Czinkoczki

KimSeong Loh

Brent Dayley

Jim Spiller

Christopher Wensley

Manish Pawar

Clair Bennett

Yanti Chang

Joel Goodman

Gerlinde Frenzen

Madhavi Siddireddy

Editors

Smita Kommini

Aju Kumar

Publishers

Pavithran Adka

Giri Venugopal

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Disclaimer

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

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

Restricted Rights Notice

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

U.S. GOVERNMENT RIGHTS

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

Trademark Notice

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

1 Introduction

Lesson Objectives

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

- Define the goals of the course
- List the features of Oracle Database 12c
- Describe the salient features of Oracle Cloud
- Discuss the theoretical and physical aspects of a relational database
- Describe Oracle server's implementation of RDBMS and object relational database management system (ORDBMS)
- Identify the development environments that can be used for this course
- Describe the database and schema used in this course



Lesson Agenda

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

Course Objectives

After completing this course, you should be able to:

- Identify the major components of Oracle Database
- Retrieve row and column data from tables with the SELECT statement
- Create reports of sorted and restricted data
- Employ SQL functions to generate and retrieve customized data
- Run complex queries to retrieve data from multiple tables
- Run data manipulation language (DML) statements to update data in Oracle Database
- Run data definition language (DDL) statements to create and manage schema objects

Course Agenda

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

Course Agenda

- Day 3:
 - Using the Set Operators
 - Managing Tables Using DML Statements
 - Introduction to Data Definition Language

Appendices and Practices Used in the Course

- Appendix A: Table Descriptions
- Appendix B: Using SQL Developer
- Appendix C: Using SQL*Plus
- Appendix D: Commonly Used SQL Commands
- Activity Guide
 - Practices and Solutions
 - Additional Practices and Solutions

Lesson Agenda

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

Oracle Database 12c: Focus Areas



Infrastructure
Grids

Information
Management

Application
Development

Oracle Cloud

Oracle Database 12c



Manageability

High Availability

Performance

Security

Information Integration

Oracle Fusion Middleware

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



Oracle Enterprise Manager Cloud Control

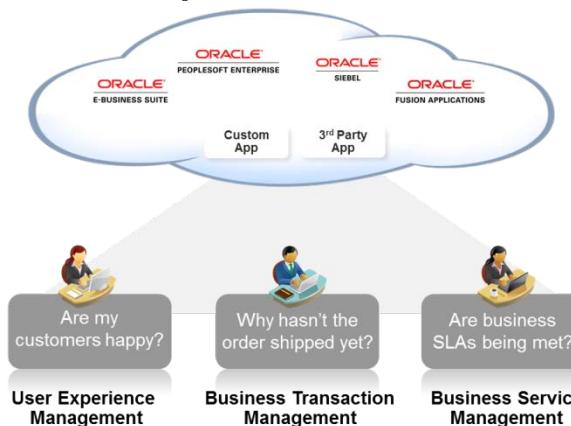
- Create and manage a complete set of cloud services.
- Manage all phases of cloud life cycle.
- Manage the entire cloud stack
- Monitor the health of all components
- Identify, understand, and resolve business problems



Complete life cycle



Complete stack



Complete integration

Self-Service IT

I Simple and Automated

I Business-Driven

Oracle Cloud

The Oracle Cloud is an enterprise cloud for business. It consists of many different services that share some common characteristics:

- On-demand self-service
- Resource pooling
- Rapid elasticity
- Measured service
- Broad network access

www.cloud.oracle.com



A screenshot of the Oracle Cloud homepage. The header features the "ORACLE CLOUD" logo. Below the header, a main banner has a blue background with a white cloud graphic. The text in the banner reads "Larry Ellison tells why Oracle Cloud is the enterprise choice." and "Watch Highlights". To the right of the text is a video player showing a portrait of Larry Ellison speaking. At the bottom of the page, there is a blue footer bar with the text "Transform your business with Oracle Cloud" and "Try it". There are also "Chat Now" and "Contact Us" links, along with the Oracle logo.

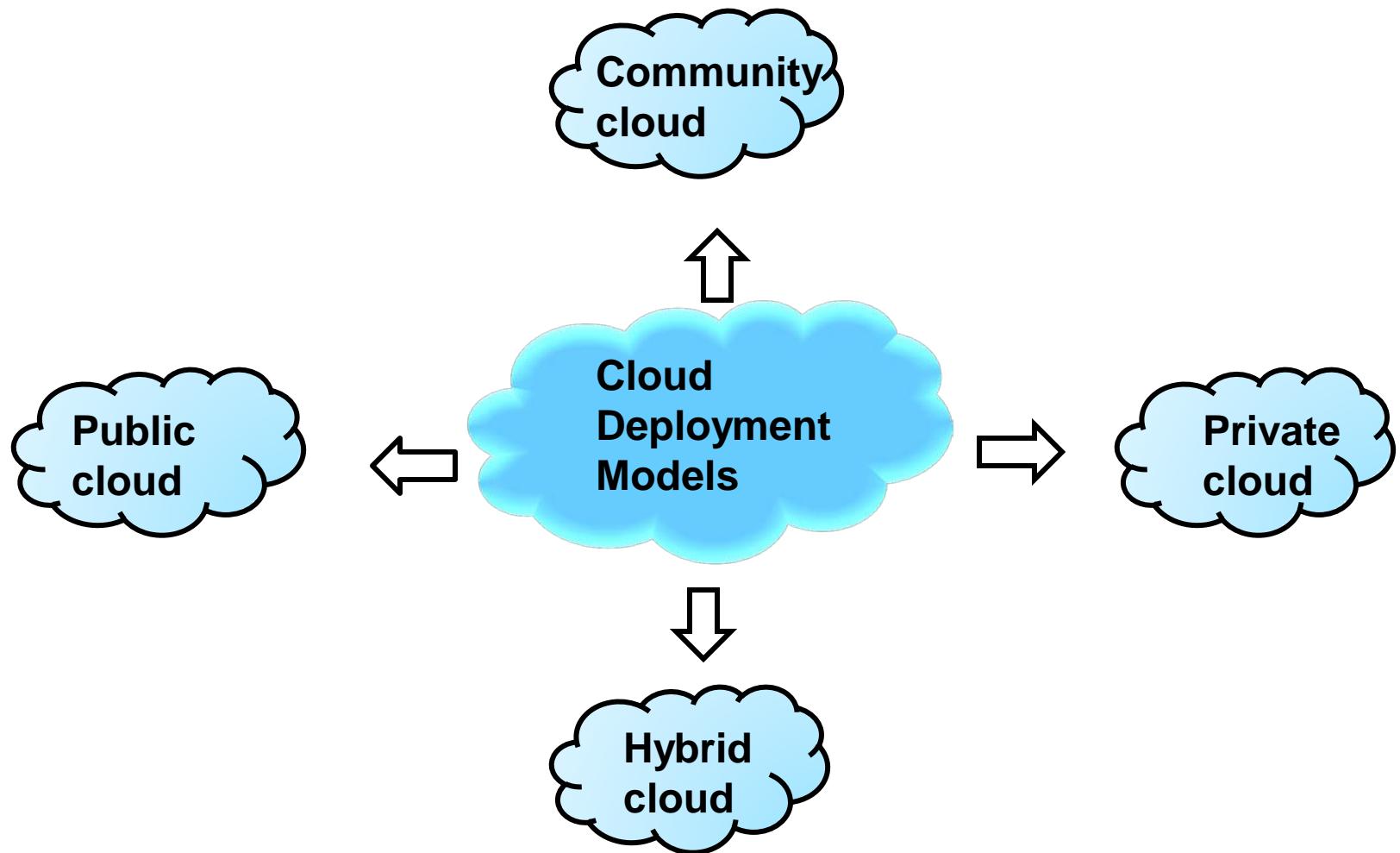
Oracle Cloud Services

Oracle Cloud provides three types of services:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)



Cloud Deployment Models



Lesson Agenda

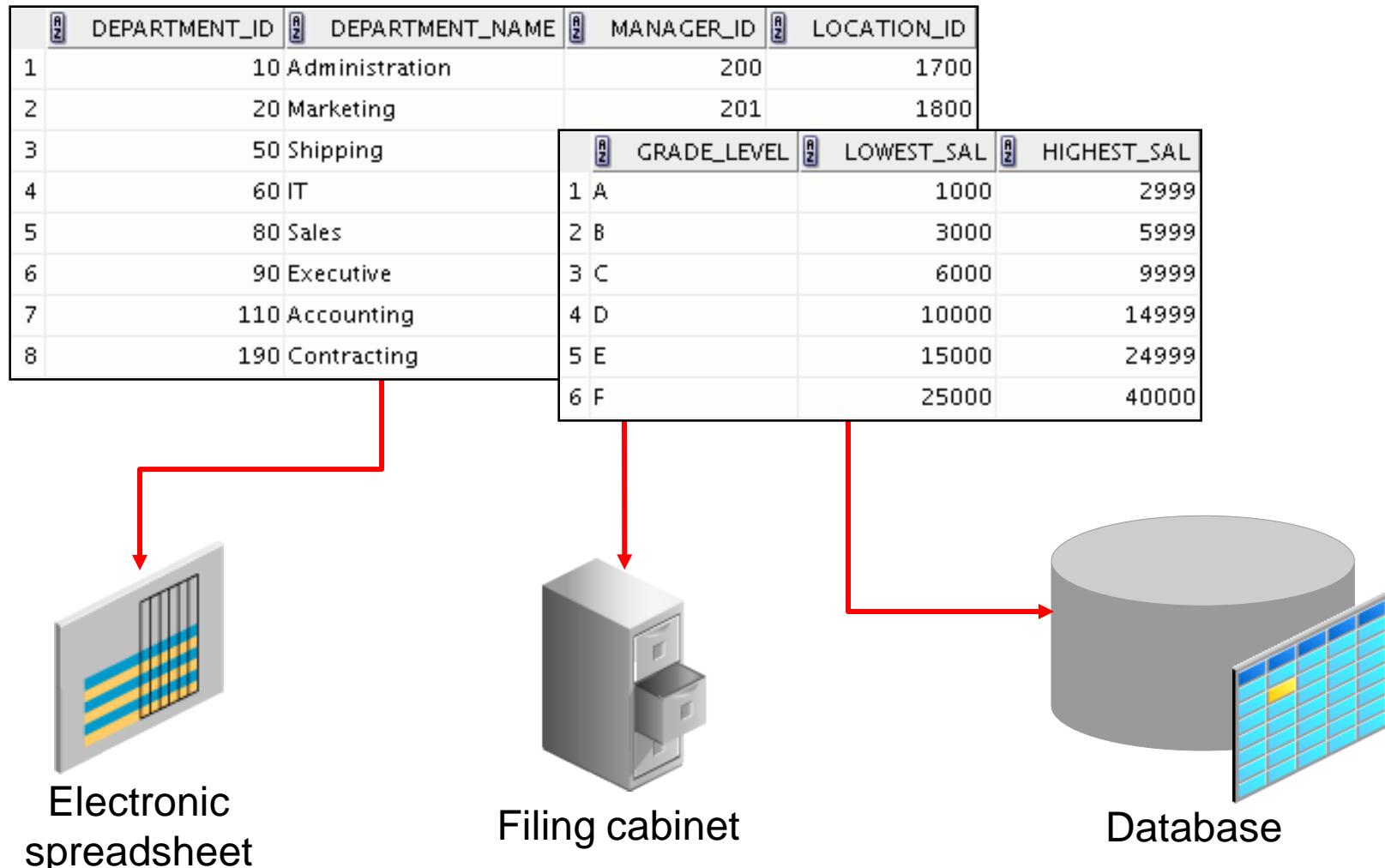
- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Database 12c and related products
- **Overview of relational database management concepts and terminologies**
- Introduction to SQL and its development environments
- Human Resource (_{HR}) Schema and the tables used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

Relational and Object Relational Database Management Systems

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



Data Storage on Different Media

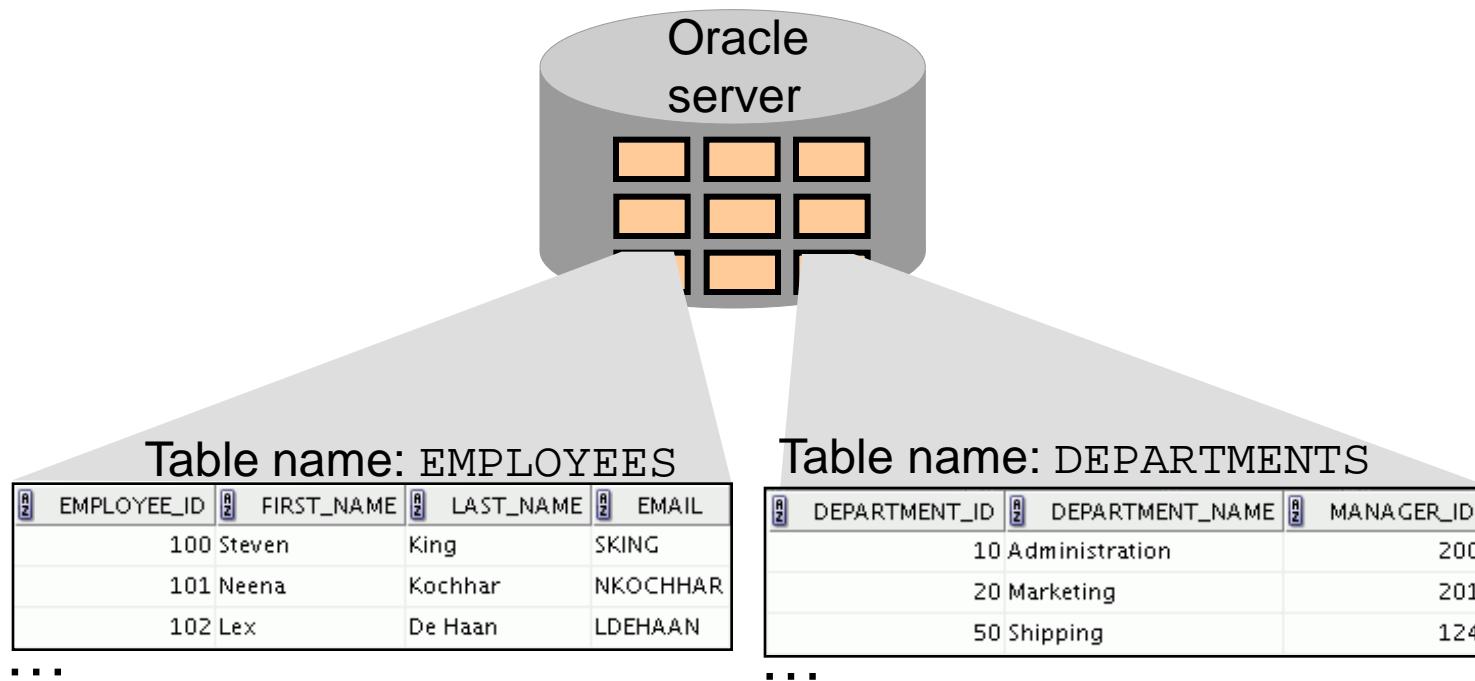


Relational Database Concept

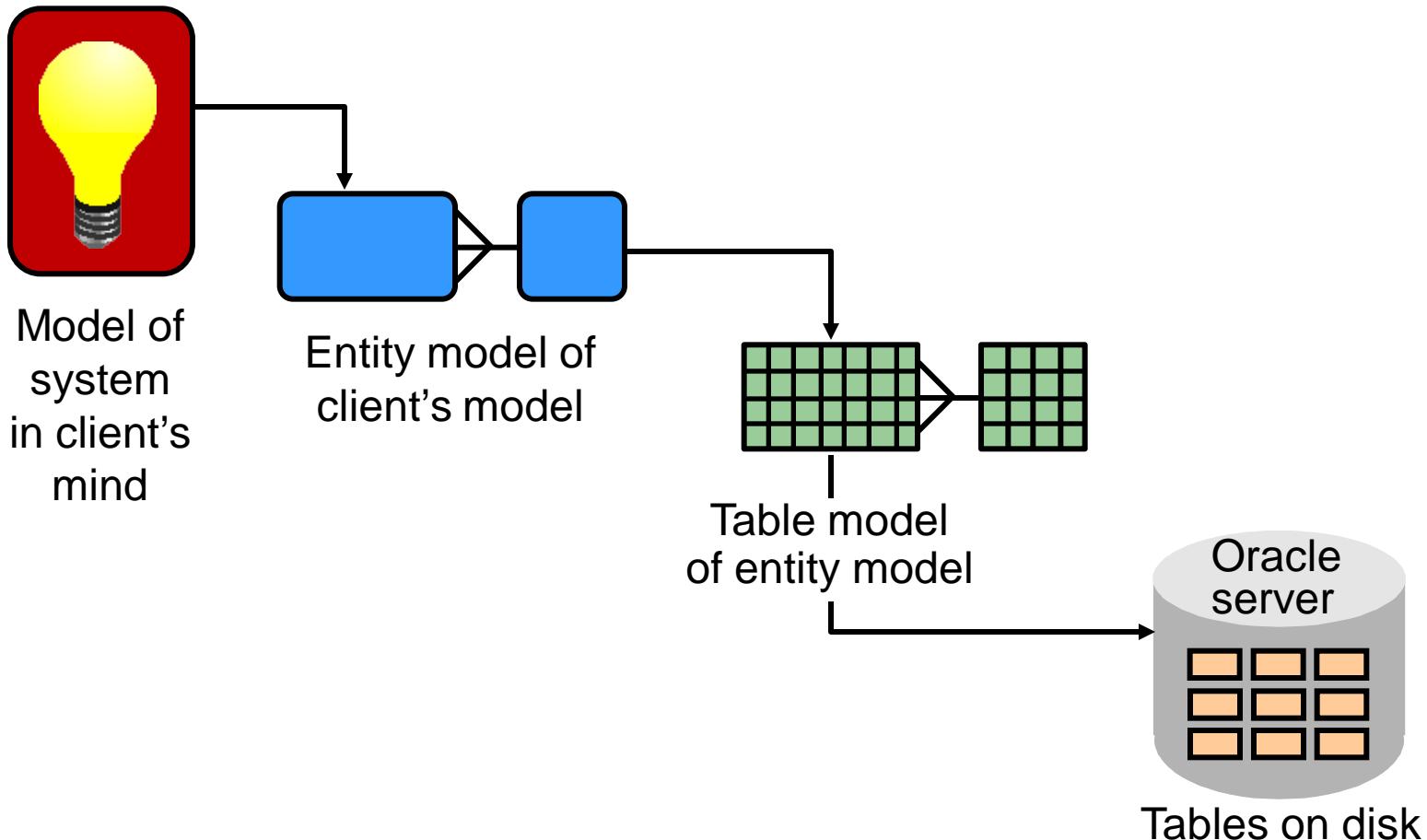
- Dr. E. F. Codd proposed the relational model for database systems in 1970.
- It is the basis for the relational database management system (RDBMS).
- The relational model consists of the following:
 - Collection of objects or relations
 - Set of operators to act on the relations
 - Data integrity for accuracy and consistency

Definition of a Relational Database

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

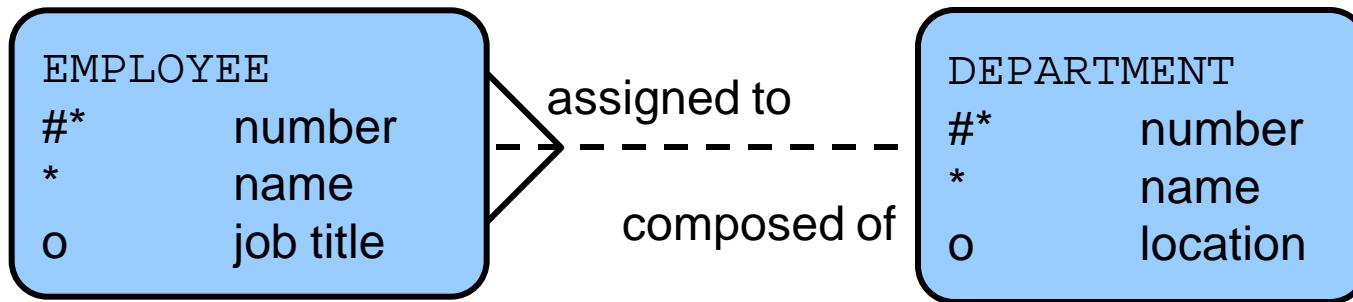


Data Models



Entity Relationship Model

- Create an entity relationship diagram from business specifications or narratives:



- Scenario:
 - "... Assign one or more employees to a department ..."
 - "... Some departments do not yet have assigned employees
..."

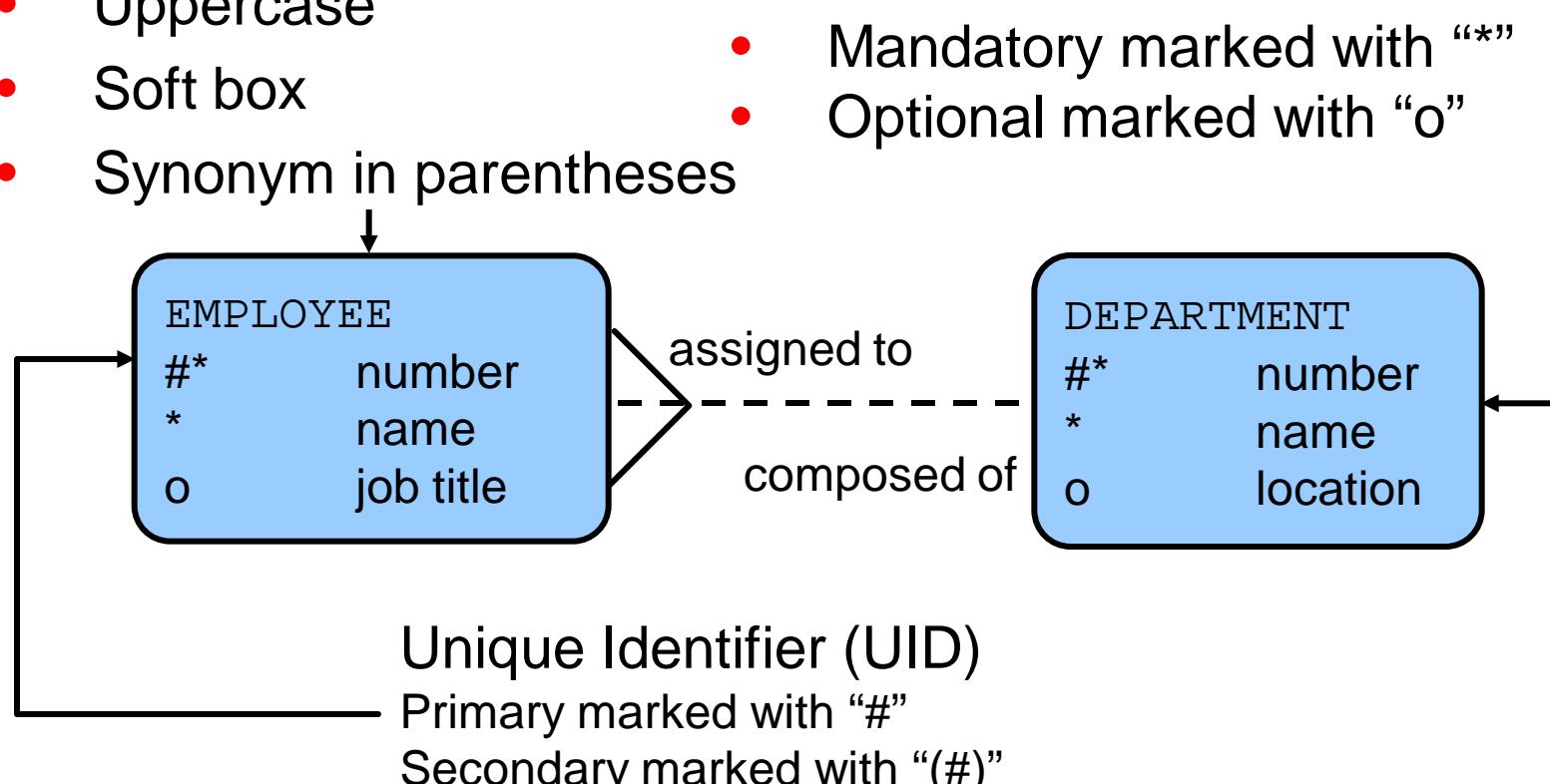
Entity Relationship Modeling Conventions

Entity:

- Singular, unique name
- Uppercase
- Soft box
- Synonym in parentheses

Attribute:

- Singular name
- Lowercase
- Mandatory marked with “*”
- Optional marked with “o”



Relating Multiple Tables

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

Table name: EMPLOYEES

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

Primary key

Foreign key

Table name: DEPARTMENTS

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

Primary key



Relational Database Terminology

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

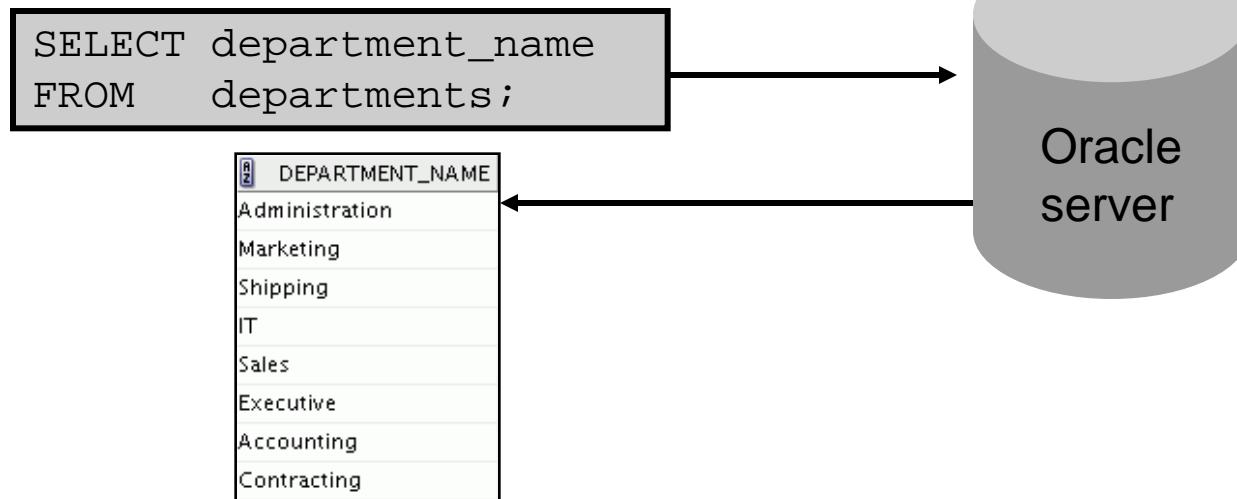
Lesson Agenda

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

Using SQL to Query Your Database

Structured query language (SQL) is:

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



SQL Statements Used in the Course

SELECT
INSERT
UPDATE
DELETE
MERGE

Data manipulation language (DML)

CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT

Data definition language (DDL)

GRANT
REVOKE

Data control language (DCL)

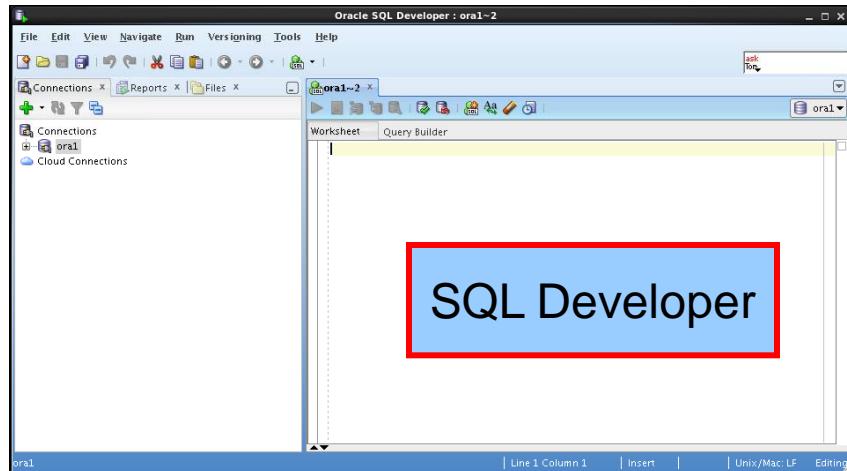
COMMIT
ROLLBACK
SAVEPOINT

Transaction control

Development Environments for SQL

There are two development environments for this course:

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

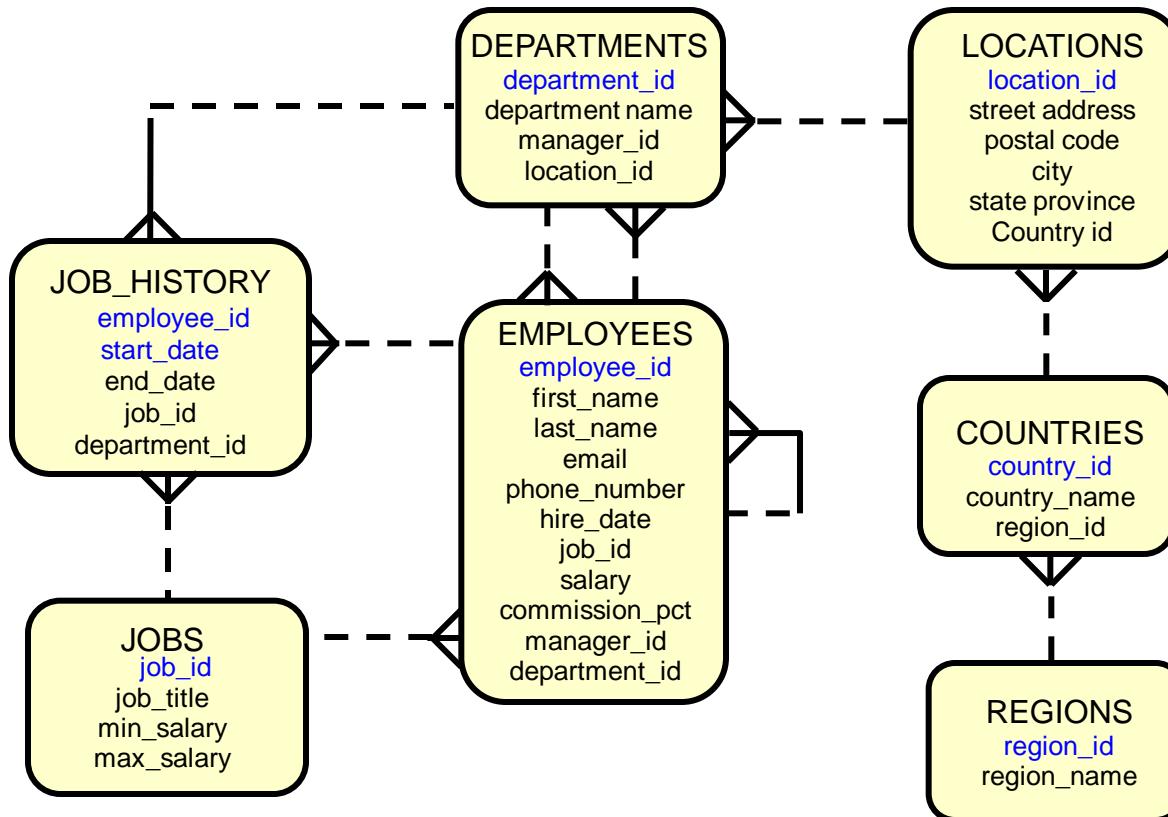


A screenshot of a terminal window titled "oracle@EDRSR25P1:~/Desktop". The window shows the SQL*Plus command-line interface. The text output includes: "File Edit View Search Terminal Help", "[oracle@EDRSR25P1 Desktop]\$ sqlplus", "SQL*Plus: Release 12.1.0.0.2 Beta on Tue Aug 28 02:06:39 2012", "Copyright (c) 1982, 2012, Oracle. All rights reserved.", and "Enter user-name:". A red box highlights the word "SQL*Plus" in the title bar.

Lesson Agenda

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

Human Resources (HR) Schema



Tables Used in the Course

EMPLOYEES

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

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

JOB_GRADES

#	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

DEPARTMENTS

Lesson Agenda

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

Oracle Database Documentation

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

Additional Resources

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

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



Summary

In this lesson, you should have learned:

- The goals of the course
- Features of Oracle Database 12c
- The salient features of Oracle Cloud
- The theoretical and physical aspects of a relational database
- Oracle server's implementation of RDBMS and object relational database management system (ORDBMS)
- The development environments that can be used for this course
- About the database and schema used in this course

Practice 1: Overview

This practice covers the following topics:

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



Retrieving Data Using the SQL SELECT Statement

Objectives

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

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

Lesson Agenda

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

Basic SELECT Statement

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

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

Selecting All Columns

```
SELECT *  
FROM departments;
```

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

Selecting Specific Columns

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

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

Writing SQL Statements

- SQL statements are not case-sensitive.
- SQL statements can be entered on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.
- In SQL Developer, SQL statements can be optionally terminated by a semicolon (;). Semicolons are required when you execute multiple SQL statements.
- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

Column Heading Defaults

- SQL Developer:
 - Default heading alignment: Left-aligned
 - Default heading display: Uppercase
- SQL*Plus:
 - Character and Date column headings are left-aligned.
 - Number column headings are right-aligned.
 - Default heading display: Uppercase

Lesson Agenda

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

Arithmetic Expressions

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

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

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

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

...

ORACLE®

Operator Precedence

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

1

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

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

2

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

Defining a Null Value

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

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

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

12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
15	Grant	SA_REP	7000	0.15

18	Fay	MK_REP	6000	(null)
19	Higgins	AC_MGR	12008	(null)
20	Gietz	AC_ACCOUNT	8300	(null)



Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

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

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

...

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

...

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

Lesson Agenda

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

Defining a Column Alias

A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name (there can also be the optional AS keyword between the column name and the alias)
- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

Using Column Aliases

```
SELECT last_name AS name, commission_pct comm  
FROM employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)
4	Hunold	(null)

...

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

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

...

Lesson Agenda

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

Concatenation Operator

A concatenation operator:

- Links columns or character strings to other columns
- Is represented by two vertical bars (| |)
- Creates a resultant column that is a character expression

```
SELECT      last_name || job_id AS "Employees"  
FROM        employees;
```

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

...

ORACLE®

Literal Character Strings

- A literal is a character, a number, or a date that is included in the SELECT statement.
- Date and character literal values must be enclosed within single quotation marks.
- Each character string is output once for each row returned.

Using Literal Character Strings

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

Employee Details	
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
6	Gietz is a AC_ACCOUNT
7	Grant is a SA_REP
8	Hartstein is a MK_MAN
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG
11	King is a AD_PRES
...	

Alternative Quote (q) Operator

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

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

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

Duplicate Rows

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

1

```
SELECT department_id  
FROM employees;
```

DEPARTMENT_ID
1
2
3
4
5
6
7
8
...

2

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

DEPARTMENT_ID
1
2
3
4
5
6
7
8

Lesson Agenda

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

Displaying Table Structure

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

```
DESC[RIBE] tablename
```

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections tree displays a connection named 'myconnection' with a selected table 'DEPARTMENTS'. The main window has a tab bar with 'Columns' highlighted in red. Below the tab bar is an 'Actions...' button. The main area is a grid showing the columns of the 'DEPARTMENTS' table:

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

Using the DESCRIBE Command

```
DESCRIBE employees
```

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8, 2)
COMMISSION_PCT		NUMBER(2, 2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Quiz

Identify the SELECT statements that execute successfully.

- a.

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

- b.

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

- c.

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

- d.

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

Summary

In this lesson, you should have learned how to write a SELECT statement that:

- Returns all rows and columns from a table
- Returns specified columns from a table
- Uses column aliases to display more descriptive column headings

Practice 2: Overview

This practice covers the following topics:

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

Restricting and Sorting Data

Objectives

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

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query
- Use ampersand substitution to restrict and sort output at run time

Lesson Agenda

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

Limiting Rows by Using a Selection

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

“retrieve all
employees in
department 90”



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

Limiting Rows That Are Selected

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

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

- The WHERE clause follows the FROM clause.

Using the WHERE Clause

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

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

Character Strings and Dates

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

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

LAST_NAME	JOB_ID	DEPARTMENT_ID
Whalen	AD_ASST	10

```
SELECT last_name  
FROM employees  
WHERE hire_date = '17-OCT-03' ;
```

LAST_NAME
Rajs

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN ...AND...	Between two values (inclusive)
IN(set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Using Comparison Operators

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

Range Conditions Using the BETWEEN Operator

Use the BETWEEN operator to display rows based on a range of values:

```
SELECT last_name, salary  
FROM employees  
WHERE salary BETWEEN 2500 AND 3500 ;
```

Lower limit Upper limit

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

Using the IN Operator

Use the IN operator to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id  
FROM   employees  
WHERE  manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101	Kochhar	17000	100
2	102	De Haan	17000	100
3	124	Mourgos	5800	100
4	149	Zlotkey	10500	100
5	201	Hartstein	13000	100
6	200	Whalen	4400	101
7	205	Higgins	12008	101
8	202	Fay	6000	201

Pattern Matching Using the LIKE Operator

- Use the LIKE operator to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
 - % denotes zero or more characters.
 - _ denotes one character.

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

	FIRST_NAME
1	Shelley
2	Steven

Combining Wildcard Characters

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

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

LAST_NAME
Kochhar
Lorentz
Mourgos

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

Using NULL Conditions

Test for nulls with the IS NULL operator.

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

	LAST_NAME	MANAGER_ID
1	King	(null)

Defining Conditions Using Logical Operators

Operator	Meaning
AND	Returns TRUE if <i>both</i> component conditions are true
OR	Returns TRUE if <i>either</i> component condition is true
NOT	Returns TRUE if the condition is false

Using the AND Operator

AND requires both the component conditions to be true:

```
SELECT employee_id, last_name, job_id, salary  
FROM   employees  
WHERE  salary >= 10000  
AND    job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149	Zlotkey	SA_MAN	10500
2	201	Hartstein	MK_MAN	13000

Using the OR Operator

OR requires either component condition to be true:

```
SELECT employee_id, last_name, job_id, salary  
FROM   employees  
WHERE  salary >= 10000  
OR     job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100	King	AD_PRES	24000
2	101	Kochhar	AD_VP	17000
3	102	De Haan	AD_VP	17000
4	124	Mourgos	ST_MAN	5800
5	149	Zlotkey	SA_MAN	10500
6	174	Abel	SA_REP	11000
7	201	Hartstein	MK_MAN	13000
8	205	Higgins	AC_MGR	12008

Using the NOT Operator

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

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

Lesson Agenda

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

Rules of Precedence

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical operator
8	AND logical operator
9	OR logical operator

You can use parentheses to override rules of precedence.

Rules of Precedence

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

1

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

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

2

	LAST_NAME	DEPARTMENT_ID	SALARY
1	Zlotkey	80	10500
2	Abel	80	11000

Lesson Agenda

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

Using the ORDER BY Clause

Sort the retrieved rows with the ORDER BY clause:

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

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

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	De Haan	AD_VP		90 13-JAN-01
2	Gietz	AC_ACCOUNT		110 07-JUN-02
3	Higgins	AC_MGR		110 07-JUN-02
4	King	AD_PRES		90 17-JUN-03
5	Whalen	AD_ASST		10 17-SEP-03
6	Rajs	ST_CLERK		50 17-OCT-03

...

ORACLE®

Sorting

- Sorting in descending order:

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

1

- Sorting by column alias:

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

2

Sorting

- Sorting by using the column's numeric position:

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

3

- Sorting by multiple columns:

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

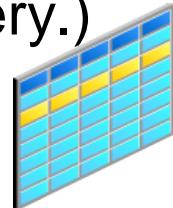
4

Lesson Agenda

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

SQL Row Limiting Clause

- You can use the `row_limiting_clause` to limit the rows that are returned by a query.
- You can use this clause to implement Top-N reporting.
- You can specify the number of rows or percentage of rows to return with the `FETCH FIRST` keyword.
- You can use the `OFFSET` keyword to specify that the returned rows begin with a row after the first row of the full result set.
- The `WITH TIES` keyword includes additional rows with the same ordering keys as the last row of the row-limited result set. (You must specify `ORDER BY` in the query.)



Using SQL Row Limiting Clause in a Query

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

Syntax:

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

SQL Row Limiting Clause: Example

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



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

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

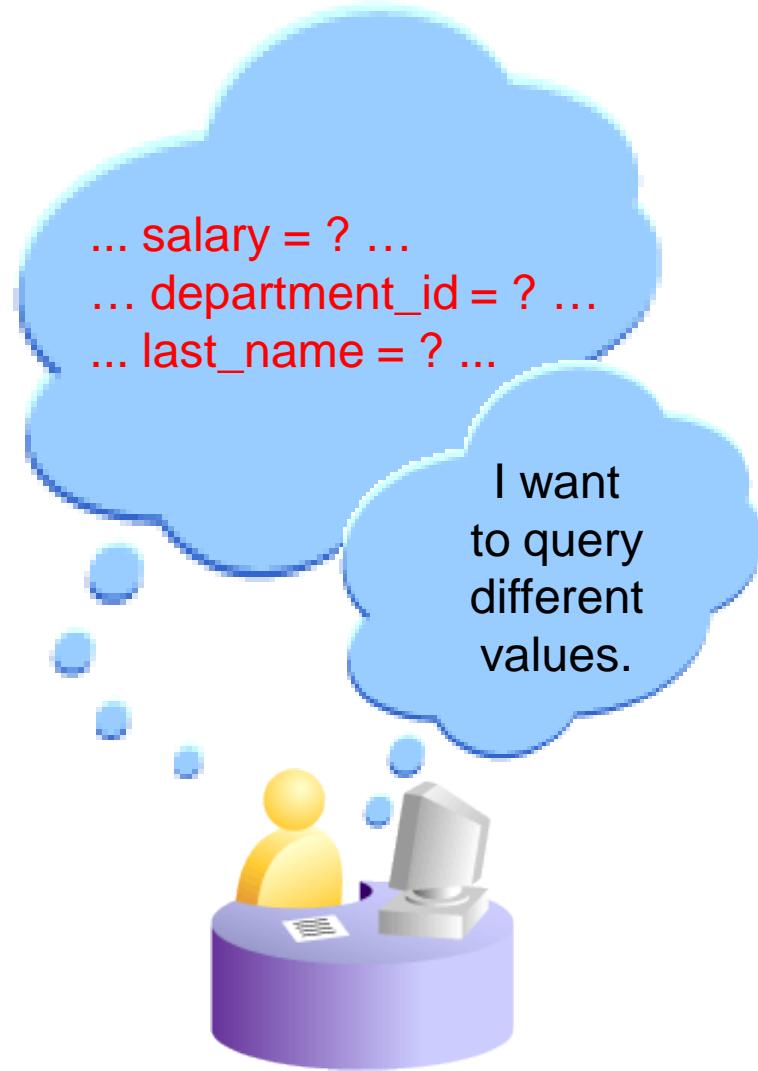


	EMPLOYEE_ID		FIRST_NAME
1	107	Diana	
2	124	Kevin	
3	141	Trenna	
4	142	Curtis	
5	143	Randall	

Lesson Agenda

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

Substitution Variables



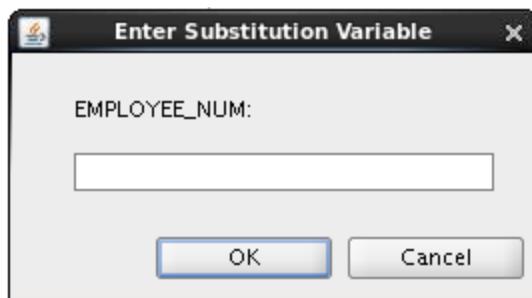
Substitution Variables

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

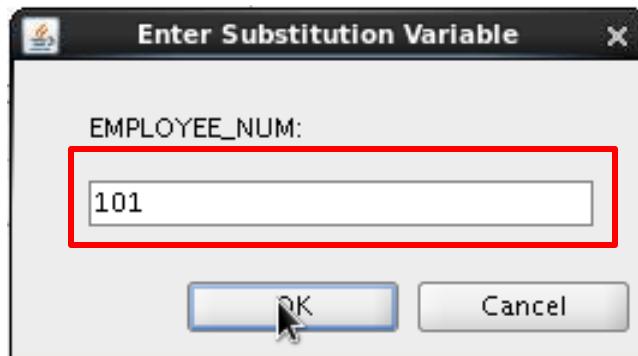
Using the Single-Ampersand Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for a value:

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



Using the Single-Ampersand Substitution Variable

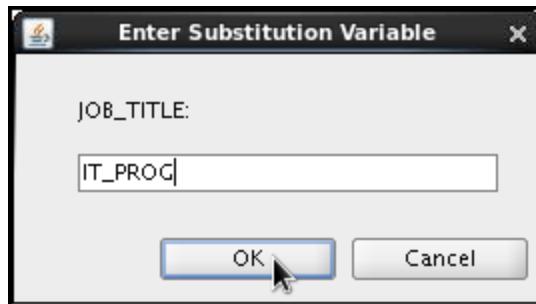


	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

Character and Date Values with Substitution Variables

Use single quotation marks for date and character values:

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

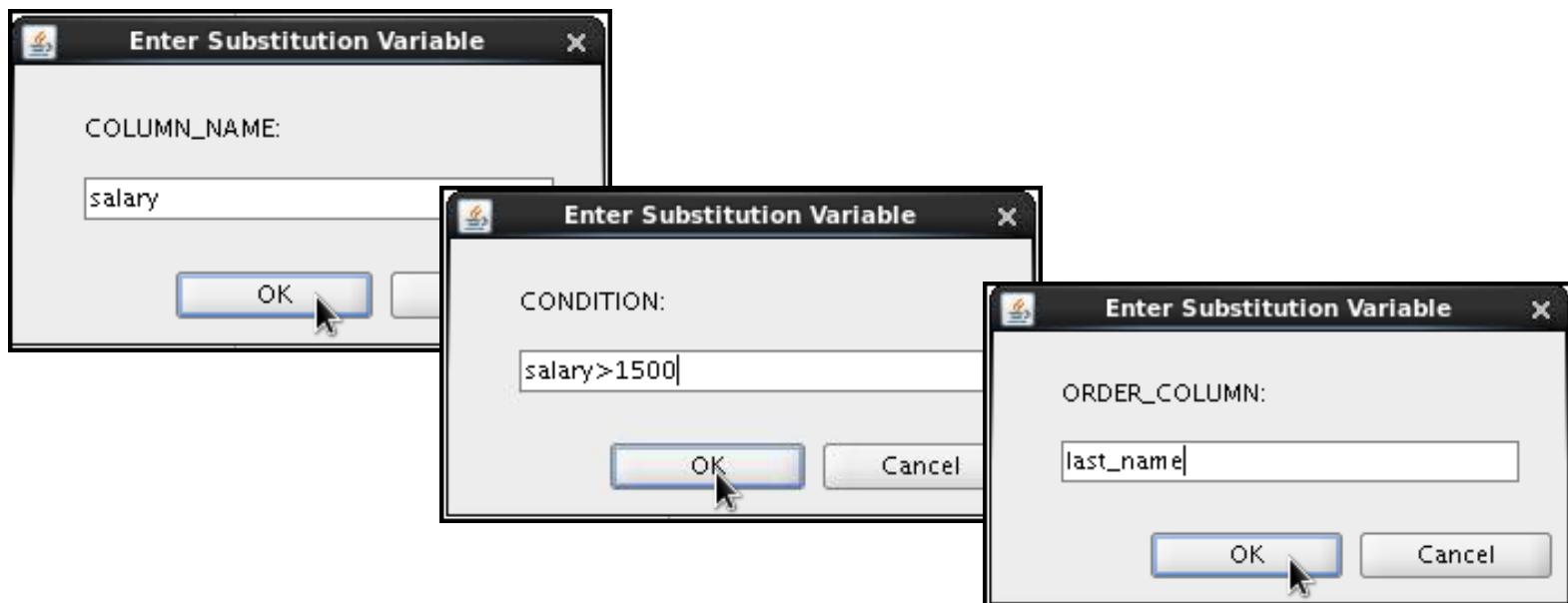


	LAST_NAME	DEPARTMENT_ID	SALARY*12
1	Hunold	60	108000
2	Ernst	60	72000
3	Lorentz	60	50400

ORACLE®

Specifying Column Names, Expressions, and Text

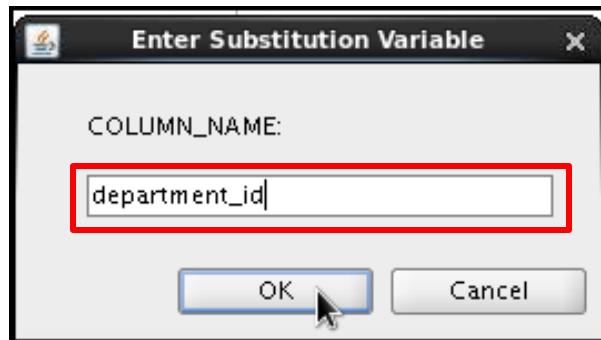
```
SELECT employee_id, last_name, job_id,&column_name  
FROM employees  
WHERE &condition  
ORDER BY &order_column ;
```



Using the Double-Ampersand Substitution Variable

Use double ampersand (`&&`) if you want to reuse the variable value without prompting the user each time:

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



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
2	201	Hartstein	MK_MAN	20
3	202	Fay	MK_REP	20

Using the Ampersand Substitution Variable in SQL*Plus

```
oracle@EDRSR19P1:~/Desktop
File Edit View Search Terminal Help
SQL> SELECT employee_id, last_name, salary, department_id
  FROM employees
 WHERE employee_id = &employee_num : 2 3
Enter value for employee_num: 101
```

```
oracle@EDRSR19P1:~/Desktop
File Edit View Search Terminal Help
SQL> SELECT employee_id, last_name, salary, department_id
  FROM employees
 WHERE employee_id = &employee_num ; 2 3
Enter value for employee_num: 101
old  3: WHERE employee_id = &employee_num
new  3: WHERE employee_id = 101
EMPLOYEE_ID LAST_NAME          SALARY DEPARTMENT_ID
----- -----
      101 Kochhar            17000          90
SQL>
```

Lesson Agenda

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

Using the DEFINE Command

- Use the DEFINE command to create and assign a value to a variable.
- Use the UNDEFINE command to remove a variable.

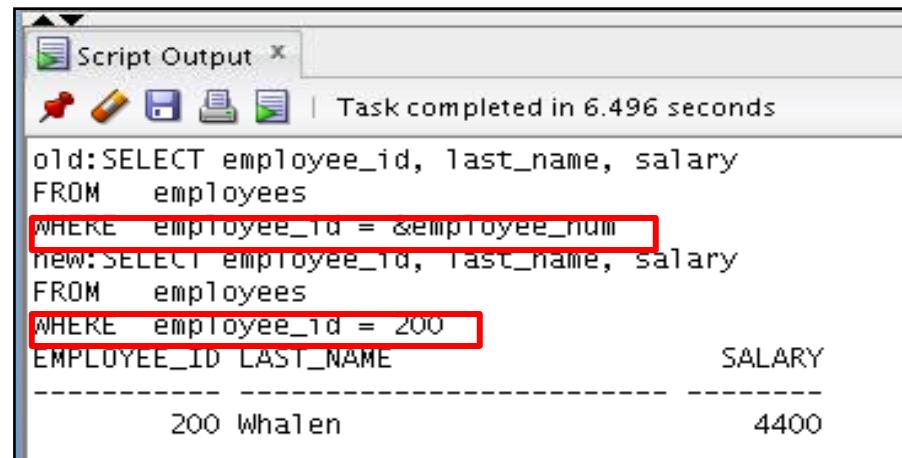
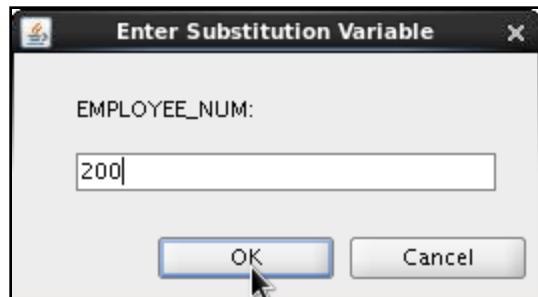
```
DEFINE employee_num = 200
      ↓
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
      ↓
UNDEFINE employee_num
```

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	200	Whalen	4400	10

Using the VERIFY Command

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

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



The "Script Output" window shows the execution of the query. The "old:" section shows the original query with the substitution variable &employee_num. The "new:" section shows the query after the variable has been replaced by the value 200. The results table shows one row for employee ID 200, last name Whalen, and salary 4400.

EMPLOYEE_ID	LAST_NAME	SALARY
200	Whalen	4400

Quiz

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

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

Summary

In this lesson, you should have learned how to:

- Limit the rows that are retrieved by a query
- Sort the rows that are retrieved by a query
- Use ampersand substitution to restrict and sort output at run time

Practice 3: Overview

This practice covers the following topics:

- Selecting data and changing the order of the rows that are displayed
- Restricting rows by using the WHERE clause
- Sorting rows by using the ORDER BY clause
- Using substitution variables to add flexibility to your SQL SELECT statements

Using Single-Row Functions to Customize Output

Objectives

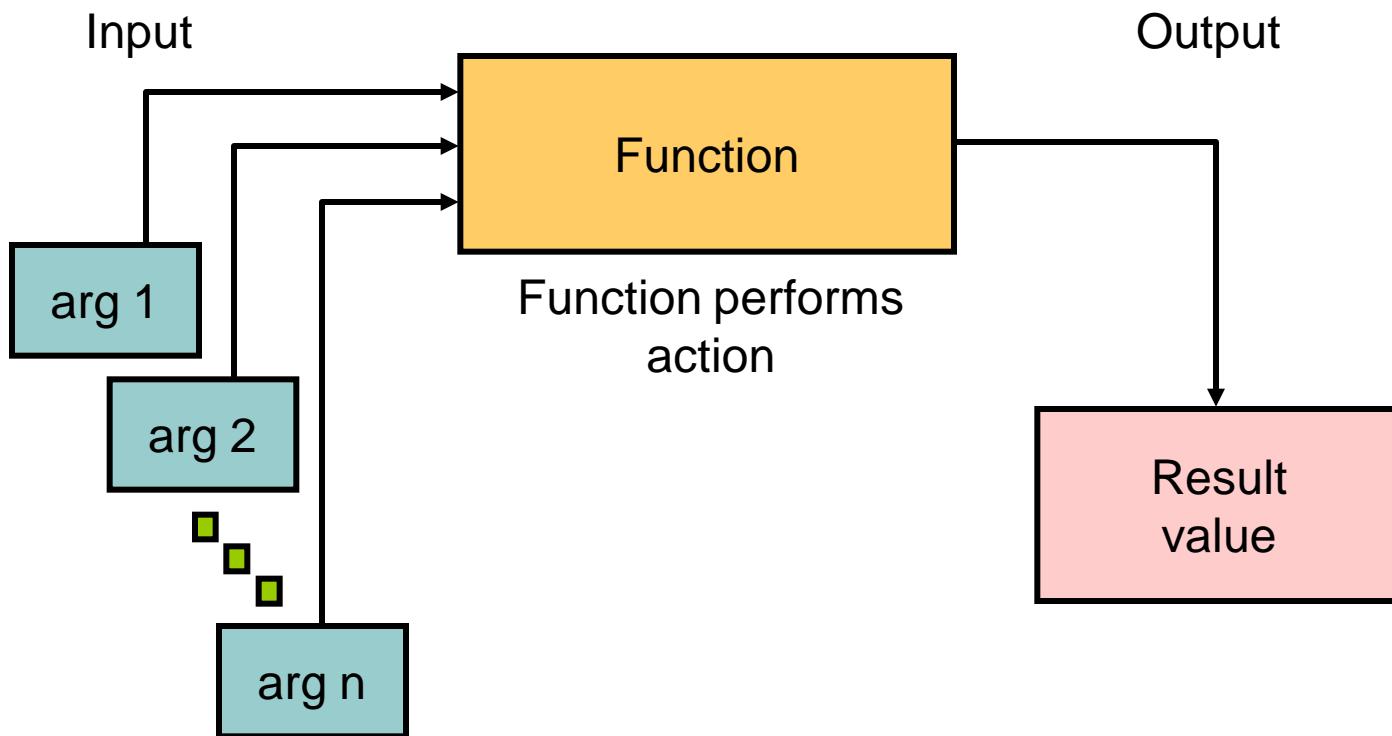
After completing this lesson, you should be able to do the following:

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

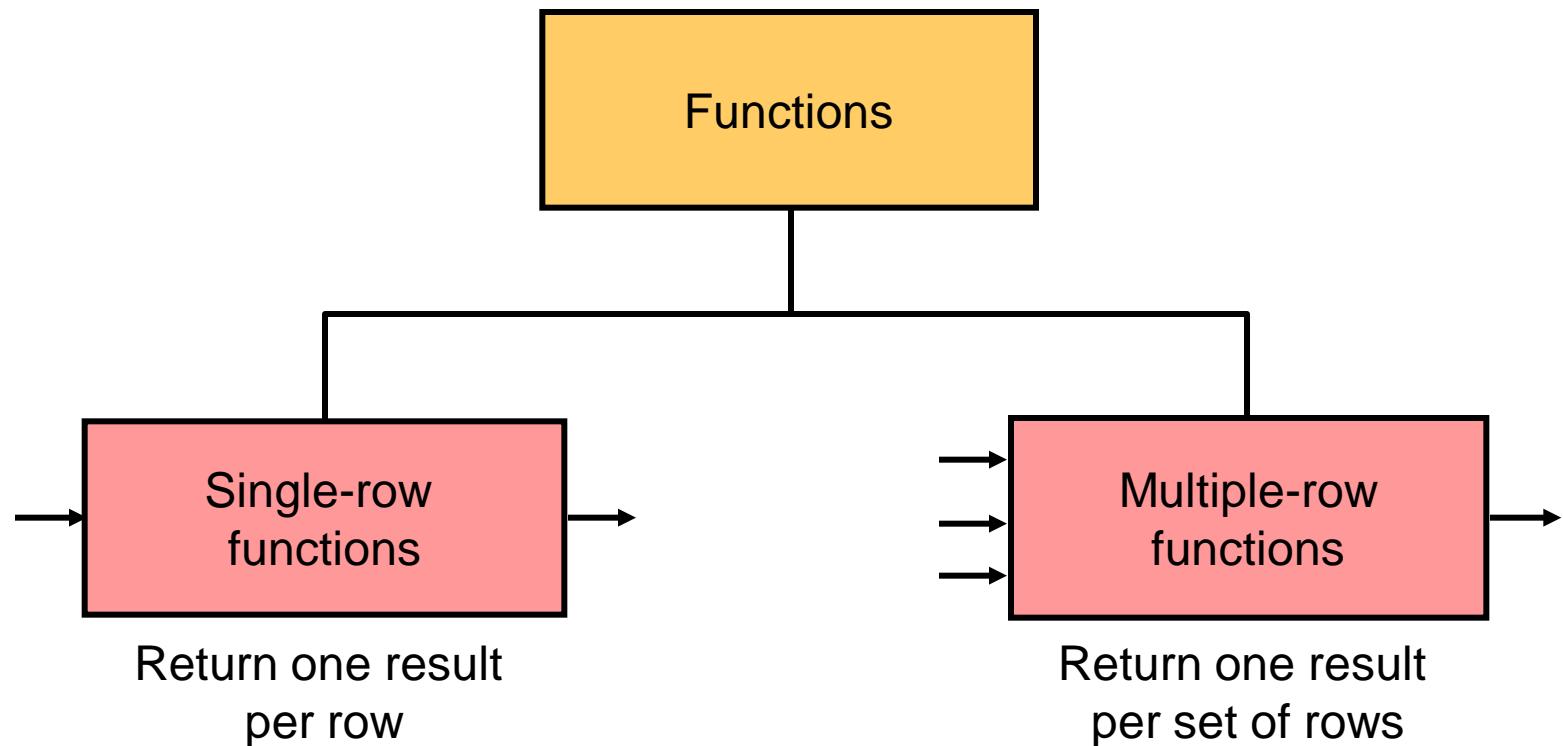
Lesson Agenda

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

SQL Functions



Two Types of SQL Functions



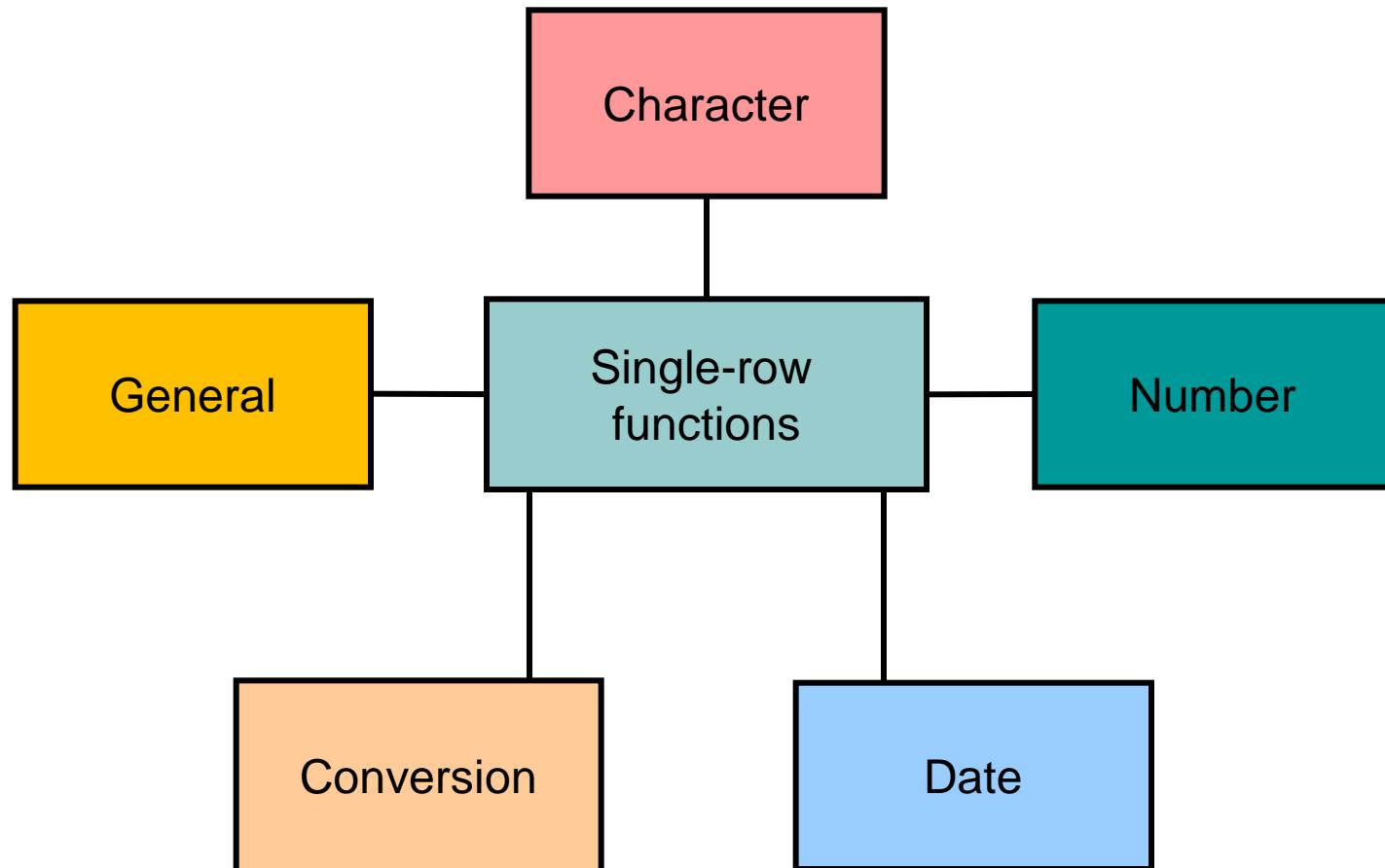
Single-Row Functions

Single-row functions:

- Manipulate data items
- Accept arguments and return one value
- Act on each row that is returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments that can be a column or an expression

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

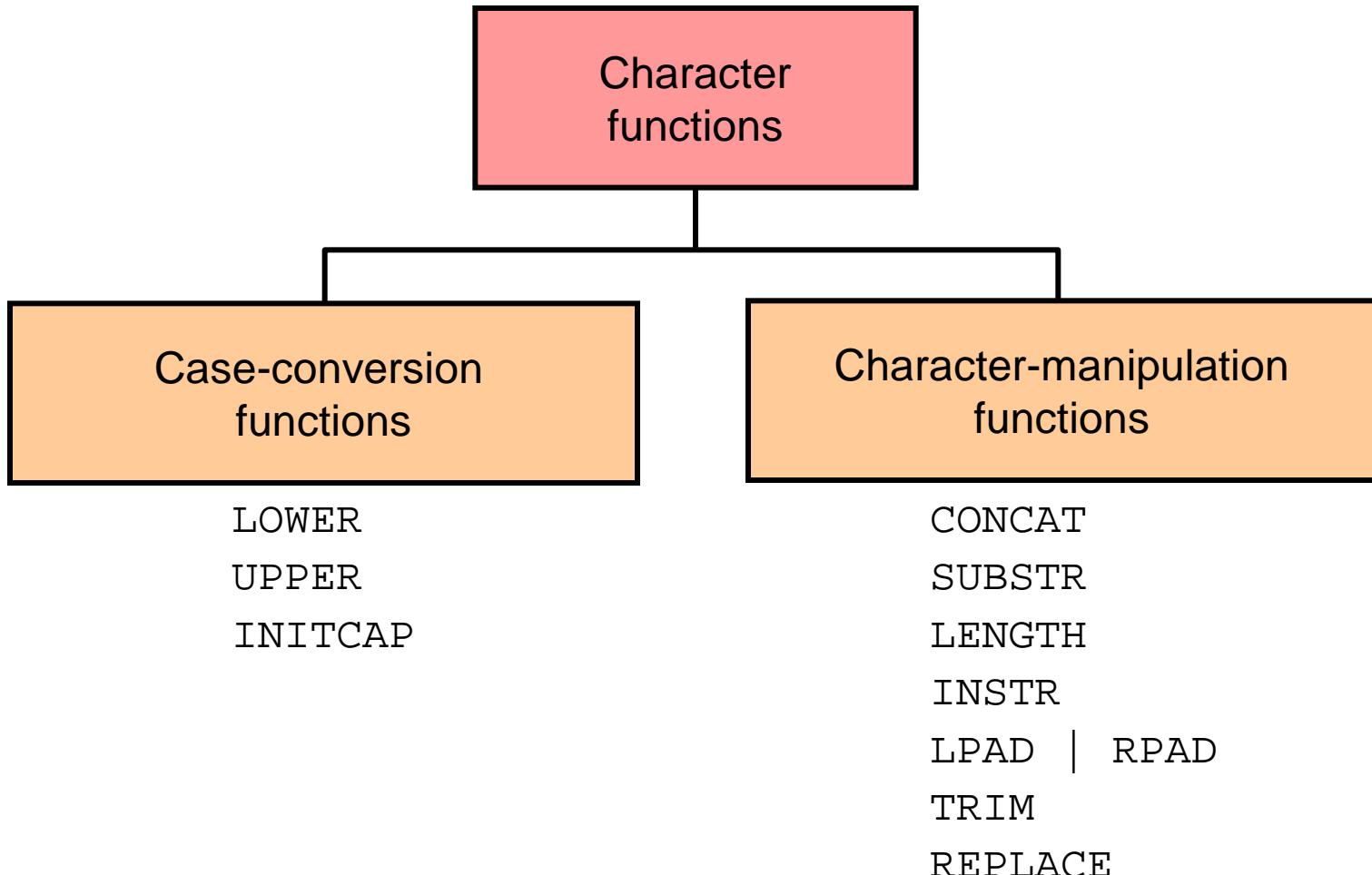
Single-Row Functions



Lesson Agenda

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

Character Functions



Case-Conversion Functions

These functions convert the case for character strings:

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

Using Case-Conversion Functions

Display the employee number, name, and department number for employee Higgins:

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

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

Character-Manipulation Functions

These functions manipulate character strings:

Function	Result
CONCAT('Hello' , 'World')	HelloWorld
SUBSTR('HelloWorld' , 1 , 5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld' , 'W')	6
LPAD(last_name , 12 , '-')	*****24000
RPAD(first_name , 12 , '-')	24000*****

Using Character-Manipulation Functions

1

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

Job
1 Abel's job category is SA_REP
2 Fay's job category is MK_REP
3 Grant's job category is SA_REP
4 Taylor's job category is SA_REP

2

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

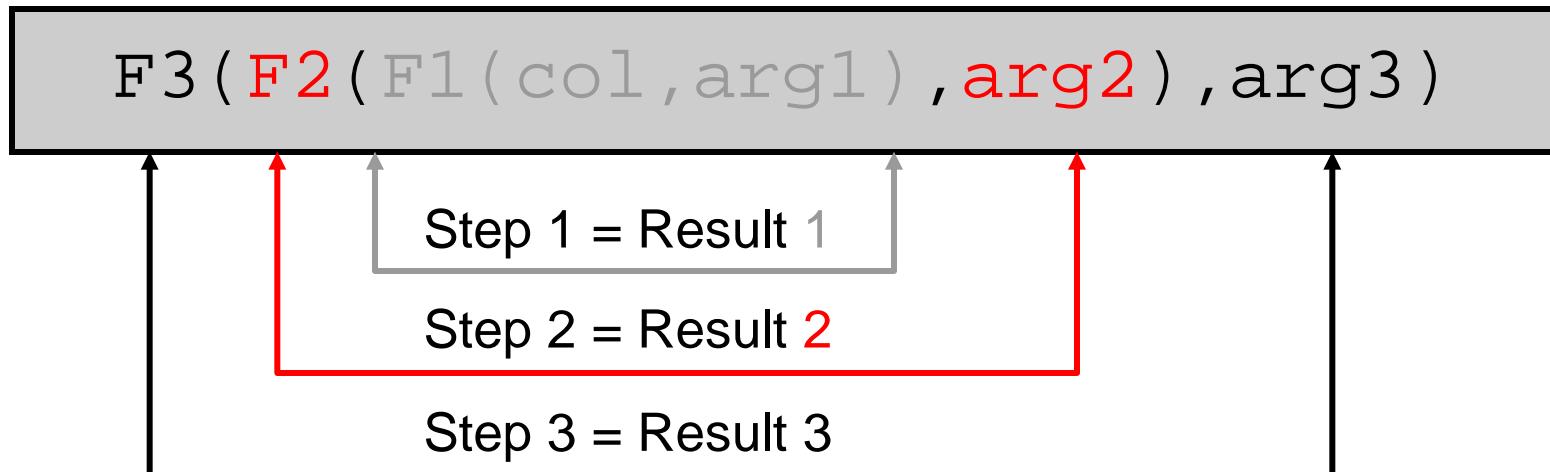
EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a?'
1	102 LexDe Haan	7	5
2	200 JenniferWhalen	6	3
3	201 MichaelHartstein	9	2

Lesson Agenda

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

Nesting Functions

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



Nesting Functions: Example

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

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

Lesson Agenda

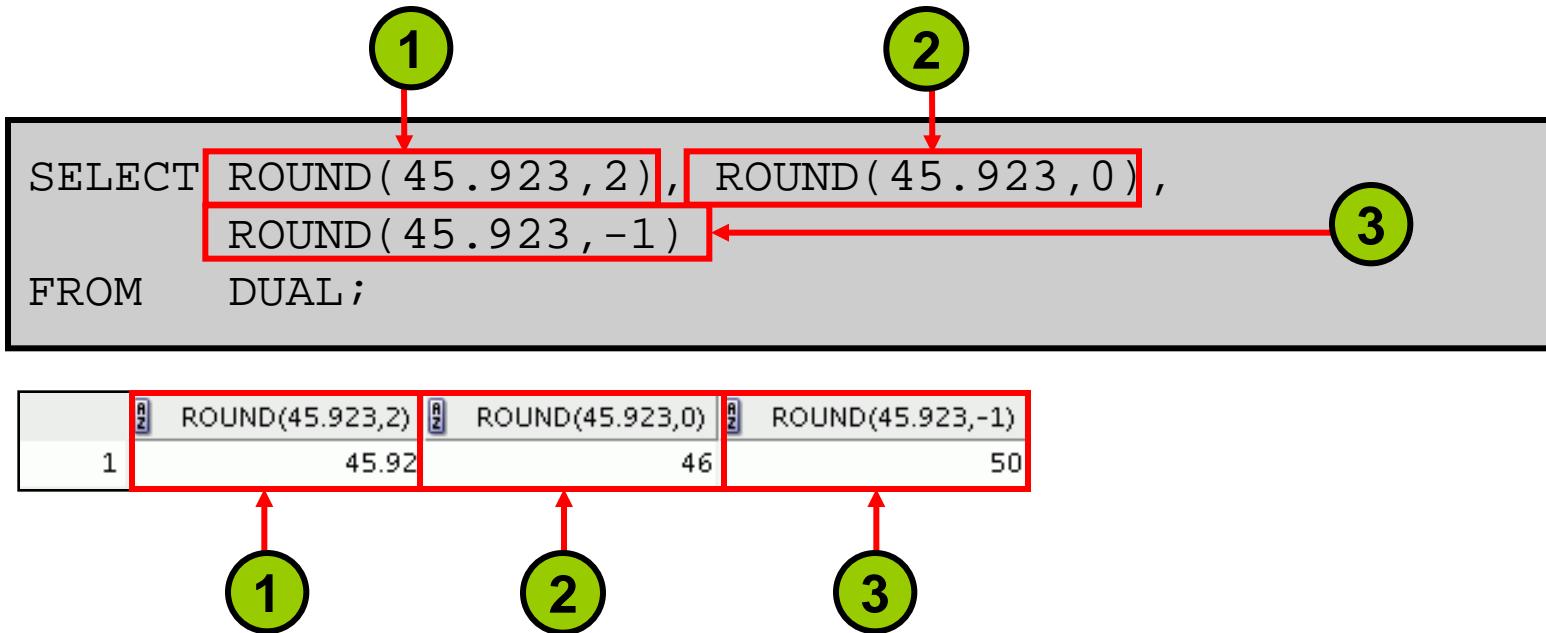
- Single-row SQL functions
- Character functions
- Nesting functions
- Number functions
- Working with dates
- Date Functions

Numeric Functions

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

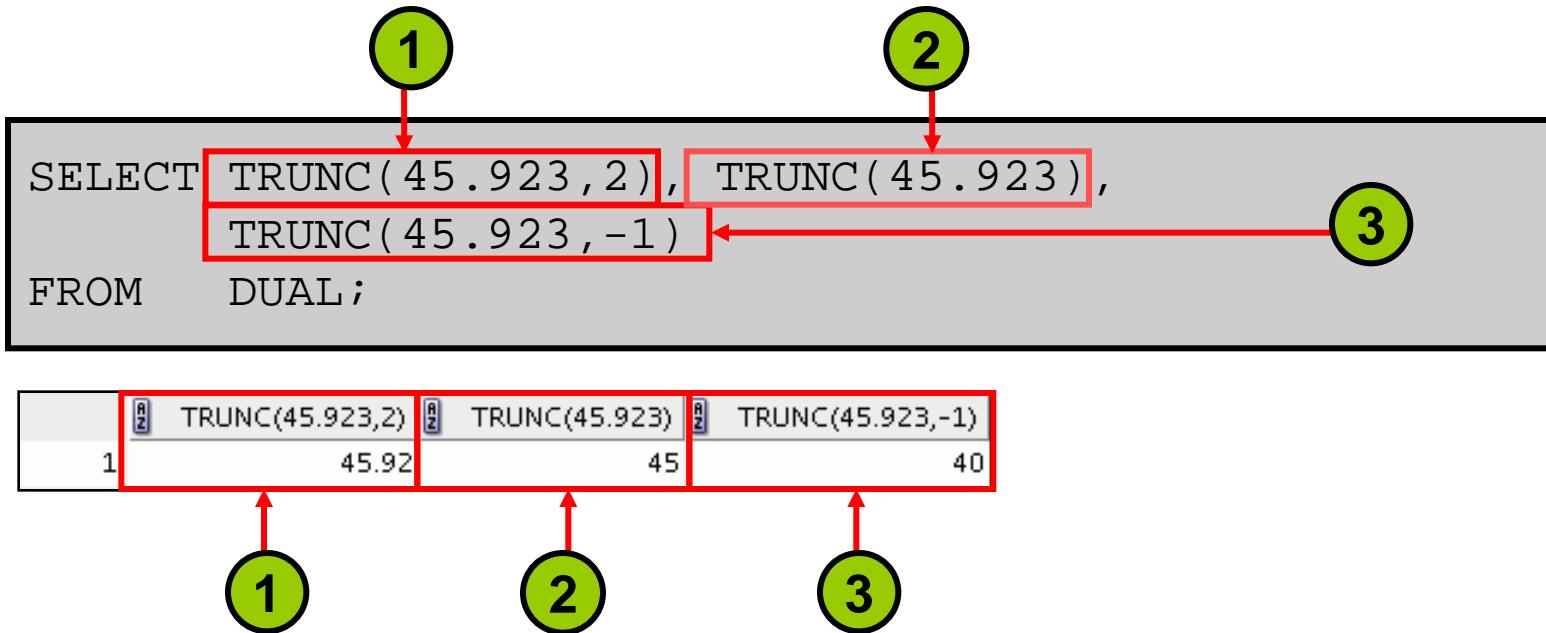
Function	Result
ROUND(45.926 , 2)	45.93
TRUNC(45.926 , 2)	45.92
CEIL (2.83)	3
FLOOR (2.83)	2
MOD (1600 , 300)	100

Using the ROUND Function



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

Using the TRUNC Function



Using the MOD Function

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

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

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

Lesson Agenda

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

Working with Dates

- The Oracle Database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.
- The default date display format is DD-MON-RR.
 - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
 - Enables you to store 20th-century dates in the 21st century in the same way

```
SELECT last_name, hire_date  
FROM employees  
WHERE hire_date < '01-FEB-2008' ;
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-03
2	Kochhar	21-SEP-05

...

ORACLE®

RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

Using the SYSDATE Function

SYSDATE is a function that returns:

- Date
- Time

```
SELECT sysdate  
FROM   dual;
```

	SYSDATE
1	24-AUG-12

Using the CURRENT_DATE and CURRENT_TIMESTAMP Functions

- CURRENT_DATE returns the current date from the user session.

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

SESSIONTIMEZONE	CURRENT_DATE
1 Etc/Universal	26-MAY-14

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

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

SESSIONTIMEZONE	CURRENT_TIMESTAMP
1 Etc/Universal	26-MAY-14 12.25.34.401622000 AM ETC/UNIVERSAL

Arithmetic with Dates

- Add to or subtract a number from a date for a resultant date value.
- Subtract two dates to find the number of days between those dates.
- Add hours to a date by dividing the number of hours by 24.

Using Arithmetic Operators with Dates

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

	LAST_NAME	WEEKS
1	King	478.871917989417989417989417989417989418
2	Kochhar	360.729060846560846560846560846560846561
3	De Haan	605.300489417989417989417989417989417989

Lesson Agenda

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

Date-Manipulation Functions

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

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-05', '11-JAN-04')	19.6774194
ADD_MONTHS ('31-JAN-04', 1)	'29-FEB-04'
NEXT_DAY ('01-SEP-05', 'FRIDAY')	'08-SEP-05'
LAST_DAY ('01-FEB-05')	'28-FEB-05'

Using ROUND and TRUNC Functions with Dates

Function	Result
ROUND(SYSDATE , 'MONTH')	01-AUG-03
ROUND(SYSDATE , 'YEAR')	01-JAN-04
TRUNC(SYSDATE , 'MONTH')	01-JUL-03
TRUNC(SYSDATE , 'YEAR')	01-JAN-03

Quiz

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

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

Summary

In this lesson, you should have learned how to:

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

Practice 4: Overview

This practice covers the following topics:

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

Using Conversion Functions and Conditional Expressions

Objectives

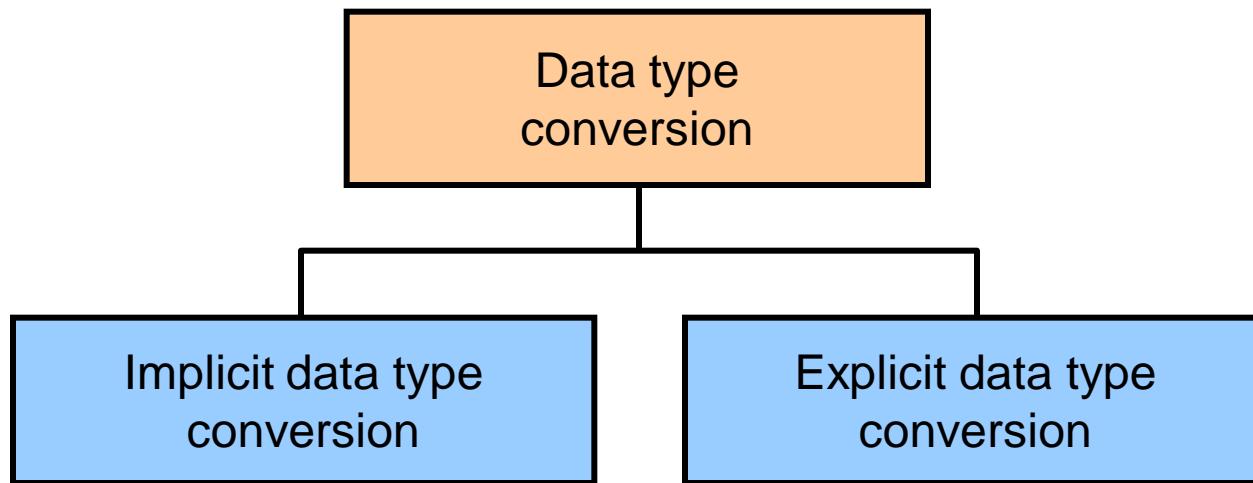
After completing this lesson, you should be able to do the following:

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

Lesson Agenda

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

Conversion Functions



Implicit Data Type Conversion

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

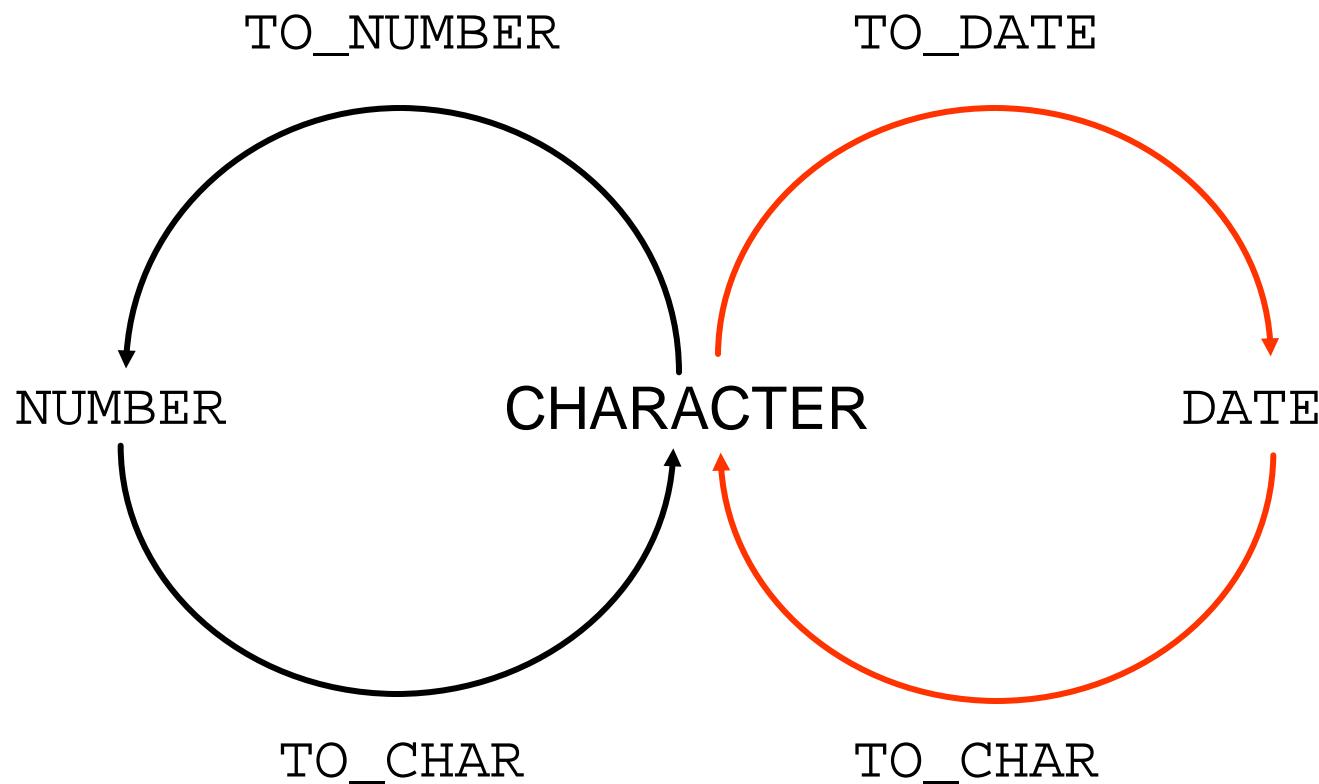
From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

Implicit Data Type Conversion

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

From	To
NUMBER	VARCHAR2 or CHAR
DATE	VARCHAR2 or CHAR

Explicit Data Type Conversion



Lesson Agenda

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

Using the TO_CHAR Function with Dates

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

The format model:

- Must be enclosed within single quotation marks
- Is case-sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for the month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

Elements of the Date Format Model

- Time elements format the time portion of the date:

HH24:MI:SS AM

15:45:32 PM

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

DD "of" MONTH

12 of OCTOBER

- Number suffixes spell out numbers:

ddspth

fourteenth

Using the TO_CHAR Function with Dates

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY')  
          AS HIREDATE  
FROM   employees;
```

	LAST_NAME	HIREDATE
1	King	17 June 2003
2	Kochhar	21 September 2005
3	De Haan	13 January 2001
4	Hunold	3 January 2006
5	Ernst	21 May 2007
6	Lorentz	7 February 2007
7	Mourgos	16 November 2007
8	Rajs	17 October 2003

Using the TO_CHAR Function with Numbers

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

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

Element	Result
9	Represents a number
0	Forces a zero to be displayed
\$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as a thousands indicator

Using the TO_CHAR Function with Numbers

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

AZ	SALARY
1	\$6,000.00

Using the TO_NUMBER and TO_DATE Functions

- Convert a character string to a number format using the TO_NUMBER function:

```
TO_NUMBER(char[ , 'format_model' ])
```

- Convert a character string to a date format using the TO_DATE function:

```
TO_DATE(char[ , 'format_model' ])
```

- These functions have an `fx` modifier. This modifier specifies the exact match for the character argument and date format model of a TO_DATE function.

Using TO_CHAR and TO_DATE Functions with the RR Date Format

To find employees hired before 1990, use the RR date format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')  
FROM employees  
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

LAST_NAME	TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
Popp	03-Feb-1989

Lesson Agenda

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

General Functions

The following functions work with any data type and pertain to using nulls:

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)
- NULLIF (expr1, expr2)
- COALESCE (expr1, expr2, . . . , exprn)

NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match:
 - `NVL(commission_pct, 0)`
 - `NVL(hire_date, '01-JAN-97')`
 - `NVL(job_id, 'No Job Yet')`

Using the NVL Function

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

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200

...



Using the NVL2 Function

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

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM



Using the NULLIF Function

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

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



Using the COALESCE Function

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

Using the COALESCE Function

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

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



Lesson Agenda

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

Conditional Expressions

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

CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
    [WHEN comparison_expr2 THEN return_expr2
     WHEN comparison_exprn THEN return_exprn
     ELSE else_expr]
END
```

Using the CASE Expression

```
SELECT last_name, job_id, salary,  
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary  
                     WHEN 'ST_CLERK' THEN 1.15*salary  
                     WHEN 'SA_REP' THEN 1.20*salary  
                     ELSE salary END "REVISED_SALARY"  
FROM employees;
```

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

Searched CASE Expression

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

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```

DECODE Function

Facilitates conditional inquiries by doing the work of a CASE expression or an IF-THEN-ELSE statement:

```
DECODE(col/expression, search1, result1
       [, search2, result2, . . . ,]
       [, default])
```

Using the DECODE Function

```
SELECT last_name, job_id, salary,  
       DECODE(job_id, 'IT_PROG', 1.10*salary,  
              'ST_CLERK', 1.15*salary,  
              'SA REP', 1.20*salary,  
              salary)  
          REVISED_SALARY  
FROM employees;
```

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

Using the DECODE Function

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

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

Quiz

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

- a. True
- b. False

Summary

In this lesson, you should have learned how to:

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

Practice 5: Overview

This practice covers the following topics:

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



Reporting Aggregated Data Using the Group Functions

Objectives

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

- Identify the available group functions
- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause

Lesson Agenda

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

Group Functions

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

EMPLOYEES

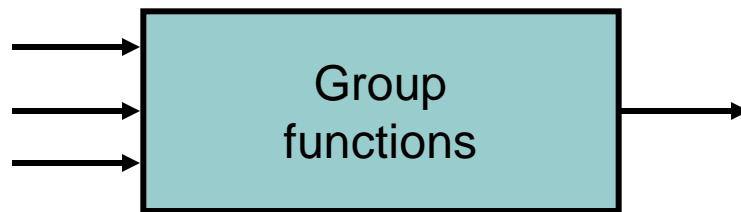
	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

Maximum salary in
EMPLOYEES table

MAX(SALARY)
24000

Types of Group Functions

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



Group Functions: Syntax

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

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%' ;
```

	Avg(Salary)	Max(Salary)	Min(Salary)	Sum(Salary)
1	8150	11000	6000	32600

Using the MIN and MAX Functions

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

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

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	13-JAN-01	29-JAN-08

Using the COUNT Function

COUNT(*) returns the number of rows in a table:

1

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

	COUNT(*)
1	5

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

2

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

	COUNT(COMMISSION_PCT)
1	0

Using the DISTINCT Keyword

- COUNT(DISTINCT expr) returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the EMPLOYEES table:

```
SELECT COUNT(DISTINCT department_id)  
FROM employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

Group Functions and Null Values

Group functions ignore null values in the column:

1

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

	Avg(COMMISSION_PCT)
1	0.2125

The NVL function forces group functions to include null values:

2

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

	Avg(NVL(COMMISSION_PCT,0))
1	0.0425

Lesson Agenda

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

Creating Groups of Data

EMPLOYEES

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

4400
9500
3500
6400
10033

Average salary in the EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	20	9500
3	90	19333.33333333333...
4	110	10150
5	50	3500
6	80	10033.33333333333...
7	10	4400
8	60	6400

Creating Groups of Data: GROUP BY Clause Syntax

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

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

Using the GROUP BY Clause

All the columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

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

Using the GROUP BY Clause

The GROUP BY column does not have to be in the SELECT list.

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

Grouping by More Than One Column

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1		10 AD_ASST	4400
2		20 MK_MAN	13000
3		20 MK_REP	6000
4		50 ST_CLERK	2500
5		50 ST_CLERK	2600
6		50 ST_CLERK	3100
7		50 ST_CLERK	3500
8		50 ST_MAN	5800
9		60 IT_PROG	9000
10		60 IT_PROG	6000
11		60 IT_PROG	4200
12		80 SA_REP	11000
13		80 SA_REP	8600
14		80 SA_MAN	10500
...			
19		110 AC_MGR	12000
20		(null) SA_REP	7000

Add the salaries in the EMPLOYEES table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1		110 AC_ACCOUNT	8300
2		110 AC_MGR	12008
3		10 AD_ASST	4400
4		90 AD_PRES	24000
5		90 AD_VP	34000
6		60 IT_PROG	19200
7		20 MK_MAN	13000
8		20 MK_REP	6000
9		80 SA_MAN	10500
10		80 SA_REP	19600
11		(null) SA_REP	7000
12		50 ST_CLERK	11700
13		50 ST_MAN	5800

Using the GROUP BY Clause on Multiple Columns

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

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

```
SELECT department_id, COUNT(last_name)  
FROM employees;
```

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

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

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

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

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

Illegal Queries Using Group Functions

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

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

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

Cannot use the
WHERE clause to
restrict groups

Restricting Group Results

EMPLOYEES

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

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

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

Restricting Group Results with the HAVING Clause

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

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

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

Using the HAVING Clause

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

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

Using the HAVING Clause

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

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

Lesson Agenda

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

Nesting Group Functions

Display the maximum average salary:

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

Quiz

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

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

Summary

In this lesson, you should have learned how to:

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

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

Practice 6: Overview

This practice covers the following topics:

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



Displaying Data from Multiple Tables Using Joins

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

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

- Write SELECT statements to access data from more than one table using equijoins and nonequijoins
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using OUTER joins
- Generate a Cartesian product of all rows from two or more tables

Lesson Agenda

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

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	100	Steven	King	AD_PRES
2	101	Neena	Kochhar	AD_VP
3	102	Lex	De Haan	AD_VP
4	103	Alexander	Hunold	IT_PROG
5	104	Bruce	Ernst	IT_PROG
6	105	David	Austin	IT_PROG
7	106	Valli	Pataballa	IT_PROG
8	107	Diana	Lorentz	IT_PROG
9	108	Nancy	Greenberg	FI_MGR
10	109	Daniel	Faviet	FI_ACCOUNT

JOBS

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

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

...

Types of Joins

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

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

Joining Tables Using SQL:1999 Syntax

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

```
SELECT      table1.column, table2.column
FROM        table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
 ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

Lesson Agenda

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

Creating Natural Joins

- The NATURAL JOIN clause is based on all the columns that have the same name in two tables.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

Retrieving Records with Natural Joins

```
SELECT employee_id, first_name, job_id, job_title  
from employees NATURAL JOIN jobs;
```

	EMPLOYEE_ID	FIRST_NAME	JOB_ID	JOB_TITLE
1	100	Steven	AD_PRES	President
2	101	Neena	AD_VP	Administration Vice President
3	102	Lex	AD_VP	Administration Vice President
4	103	Alexander	IT_PROG	Programmer
5	104	Bruce	IT_PROG	Programmer
6	105	David	IT_PROG	Programmer
7	106	Valli	IT_PROG	Programmer
8	107	Diana	IT_PROG	Programmer
9	108	Nancy	FI_MGR	Finance Manager
10	109	Daniel	FI_ACCOUNT	Accountant
11	110	John	FI_ACCOUNT	Accountant

...

ORACLE®

Creating Joins with the USING Clause

- If several columns have the same names but the data types do not match, use the USING clause to specify the columns for the equijoin.
- Use the USING clause to match only one column when more than one column matches.

Joining Column Names

EMPLOYEES

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

DEPARTMENTS

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



Retrieving Records with the USING Clause

```
SELECT employee_id, last_name,  
       location_id, department_id  
FROM   employees JOIN departments  
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50
...				
18	206	Gietz	1700	110
19	205	Higgins	1700	110



Qualifying Ambiguous Column Names

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

Using Table Aliases with the USING Clause

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

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

ORA-25154: column part of USING clause cannot have qualifier
25154. 00000 - "column part of USING clause cannot have qualifier"
*Cause: Columns that are used for a named-join (either a NATURAL join
or a join with a USING clause) cannot have an explicit qualifier.
*Action: Remove the qualifier.
Error at Line: 4 Column: 6

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

Retrieving Records with the ON Clause

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	124	Mourgos	50	50	1500
5	144	Vargas	50	50	1500
6	143	Matos	50	50	1500
7	142	Davies	50	50	1500
8	141	Rajs	50	50	1500
9	107	Lorentz	60	60	1400
10	104	Ernst	60	60	1400
11	103	Hunold	60	60	1400
...					

Creating Three-Way Joins

```
SELECT employee_id, city, department_name  
FROM employees e  
JOIN departments d  
ON d.department_id = e.department_id  
JOIN locations l  
ON d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping
...			



Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

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

Or

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

Lesson Agenda

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

Joining a Table to Itself

EMPLOYEES (WORKER)

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

...

EMPLOYEES (MANAGER)

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

...

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

Self-Joins Using the ON Clause

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

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

Lesson Agenda

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

Nonequijoins

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES

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

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

Retrieving Records with NonequiJoins

```
SELECT e.last_name, e.salary, j.grade_level  
FROM   employees e JOIN job_grades j  
ON     e.salary  
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C
...			

Lesson Agenda

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

Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

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

Equijoin with EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1		10 Whalen
2		20 Hartstein
3		20 Fay
4		110 Higgins
5		110 Gietz
6		90 King
7		90 Kochhar
8		90 De Haan
9		60 Hunold
10		60 Ernst

...

18	80 Abel
19	80 Taylor

There are no employees
in department 190.

Employee “Grant” has
not been assigned a _____
department ID.

INNER Versus OUTER Joins

- In SQL:1999, the join of two tables returning only matched rows is called an `INNER` join.
- A join between two tables that returns the results of the `INNER` join as well as the unmatched rows from the left (or right) table is called a left (or right) `OUTER` join.
- A join between two tables that returns the results of an `INNER` join as well as the results of a left and right join is a full `OUTER` join.

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e LEFT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping
...			

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name  
FROM employees e RIGHT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping
...			
18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name  
FROM employees e FULL OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	King	90	Executive
2	Kochhar	90	Executive
3	De Haan	90	Executive
4	Hunold	60	IT

...

15	Grant	(null)	(null)
16	Whalen	10	Administration
17	Hartstein	20	Marketing
18	Fay	20	Marketing
19	Higgins	110	Accounting
20	Gietz	110	Accounting
21	(null)	190	Contracting

Lesson Agenda

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

Cartesian Products

- Cartesian product is a join of every row of one table to every row of another table.
- A Cartesian product generates a large number of rows and the result is rarely useful.

Generating a Cartesian Product

EMPLOYEES (20 rows)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS (8 rows)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian product:
 $20 \times 8 = 160$ rows

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

Creating Cross Joins

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables.
- To create a Cartesian product, specify the CROSS JOIN in your SELECT statement.

```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		

158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting



Quiz

If you join a table to itself, what kind of join are you using?

- a. Nonequi join
- b. Left OUTER join
- c. Right OUTER join
- d. Full OUTER join
- e. Self-join
- f. Natural join
- g. Cartesian products

Summary

In this lesson, you should have learned how to :

- Write SELECT statements to access data from more than one table using equijoins and nonequijoins
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using OUTER joins
- Generate a Cartesian product of all rows from two or more tables

Practice 7: Overview

This practice covers the following topics:

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

Using Subqueries to Solve Queries

Objectives

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

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

Lesson Agenda

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

Using a Subquery to Solve a Problem

Who is hired after Davies?

Main query:



Determine the names of all employees who were hired after Davies?

Subquery:



When was Davies hired?

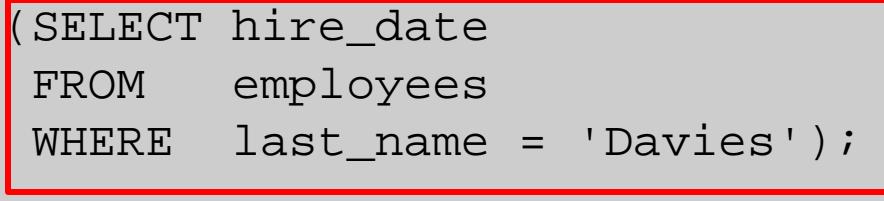
Subquery Syntax

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

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

Using a Subquery

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



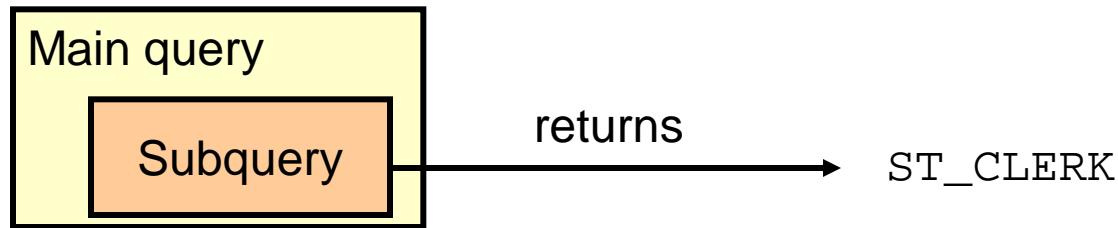
29-JAN-05 ←

Rules and Guidelines for Using Subqueries

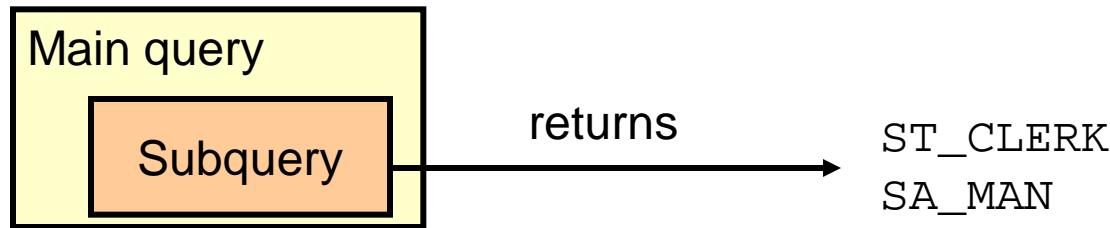
- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability. (However, the subquery can appear on either side of the comparison operator.)
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

Types of Subqueries

- Single-row subquery



- Multiple-row subquery



Lesson Agenda

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

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

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

Executing Single-Row Subqueries

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

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary  
FROM   employees  
WHERE  salary = 2500  
       (SELECT MIN(salary)  
        FROM   employees);
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

HAVING Clause with Subqueries

- The Oracle server executes the subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

```
SELECT      department_id, MIN(salary)
FROM        employees
GROUP BY    department_id
HAVING      MIN(salary) > 2500
( SELECT  MIN(salary)
  FROM    employees
  WHERE   department_id = 30 )
```

	DEPARTMENT_ID	MIN(SALARY)
1	100	6900
2	(null)	7000
3	90	17000
4	20	6000
5	70	10000
6	110	8300
7	80	6100
8	40	6500
9	60	4200
10	10	4400

ORACLE®

What Is Wrong with This Statement?

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

ORA-01427: single-row subquery returns more than one row
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:

Single-row operator with
multiple-row subquery

No Rows Returned by the Inner Query

```
SELECT last_name, job_id  
FROM   employees  
WHERE  job_id =  
       (SELECT job_id  
        FROM   employees  
        WHERE  last_name = 'Haas' );
```

Query Result	
	SQL All Rows Fetched: 0 in 0.003 seconds
LAST_N...	JOB_ID

Subquery returns no rows because there is no employee named “Haas.”

Lesson Agenda

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

Multiple-Row Subqueries

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

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Returns TRUE if at least one element exists in the result set of the subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. Returns TRUE if the relation is TRUE for all elements in the result set of the subquery.

Using the ANY Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ANY
       (SELECT salary
        FROM   employees
        WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600



Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ALL
       (SELECT salary
        FROM   employees
        WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

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

Multiple-Column Subqueries

- A multiple-column subquery returns more than one column to the outer query.
- Column comparisons in multiple column comparisons can be pairwise or nonpairwise.
- A multiple-column subquery can also be used in the FROM clause of a SELECT statement.

Multiple-Column Subquery: Example

Display all the employees with the lowest salary in each department

```
SELECT first_name, department_id, salary
FROM employees
WHERE (salary, department_id) IN
    (SELECT min(salary), department_id
     FROM employees
     GROUP BY department_id)
ORDER BY department_id;
```

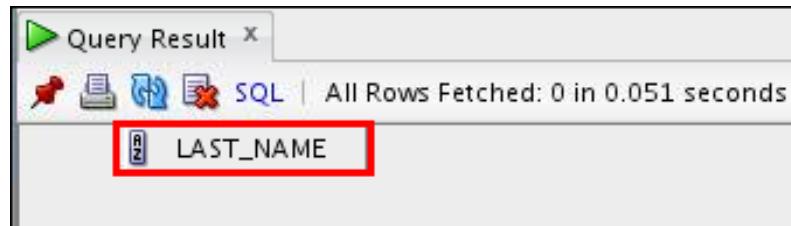
	FIRST_NAME	DEPARTMENT_ID	SALARY
1	Jennifer	10	4400
2	Pat	20	6000
3	Peter	50	2500
4	Diana	60	4200
5	Jonathon	80	8600
6	Neena	90	17000
7	Lex	90	17000
8	William	110	8300

Lesson Agenda

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

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
       (SELECT mgr.manager_id
        FROM   employees mgr);
```



Subquery returns no rows because one of the values returned by a subquery is null.

Quiz

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

- a. True
- b. False

Summary

In this lesson, you should have learned how to:

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

Practice 8: Overview

This practice covers the following topics:

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



Using the Set Operators

Objectives

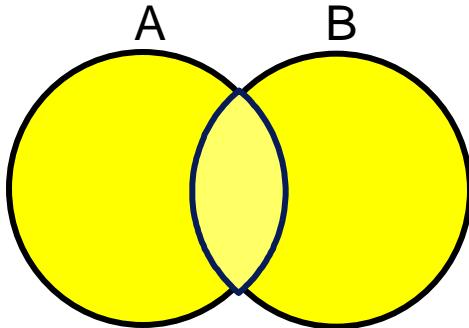
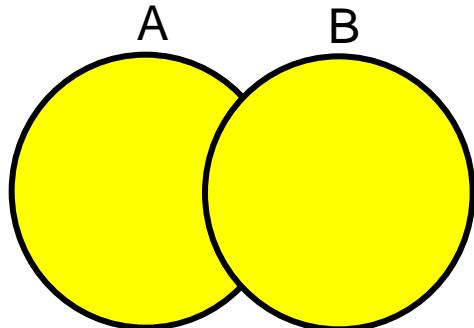
After completing this lesson, you should be able to do the following:

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

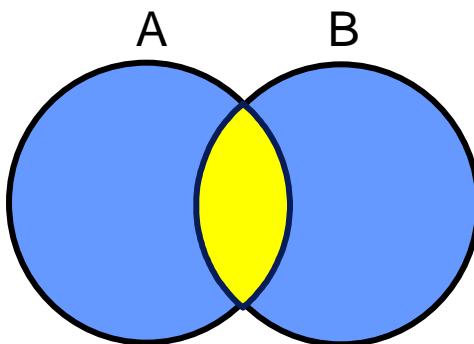
Lesson Agenda

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

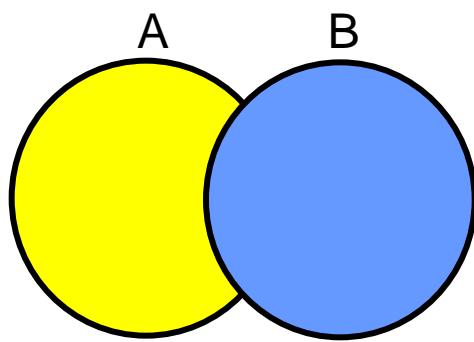
Set Operators



UNION/UNION ALL



INTERSECT



MINUS

Set Operator Rules

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

Oracle Server and Set Operators

- Duplicate rows are automatically eliminated except in UNION ALL.
- Column names from the first query appear in the result.
- The output is sorted in ascending order by default except in UNION ALL.

Lesson Agenda

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

Tables Used in This Lesson

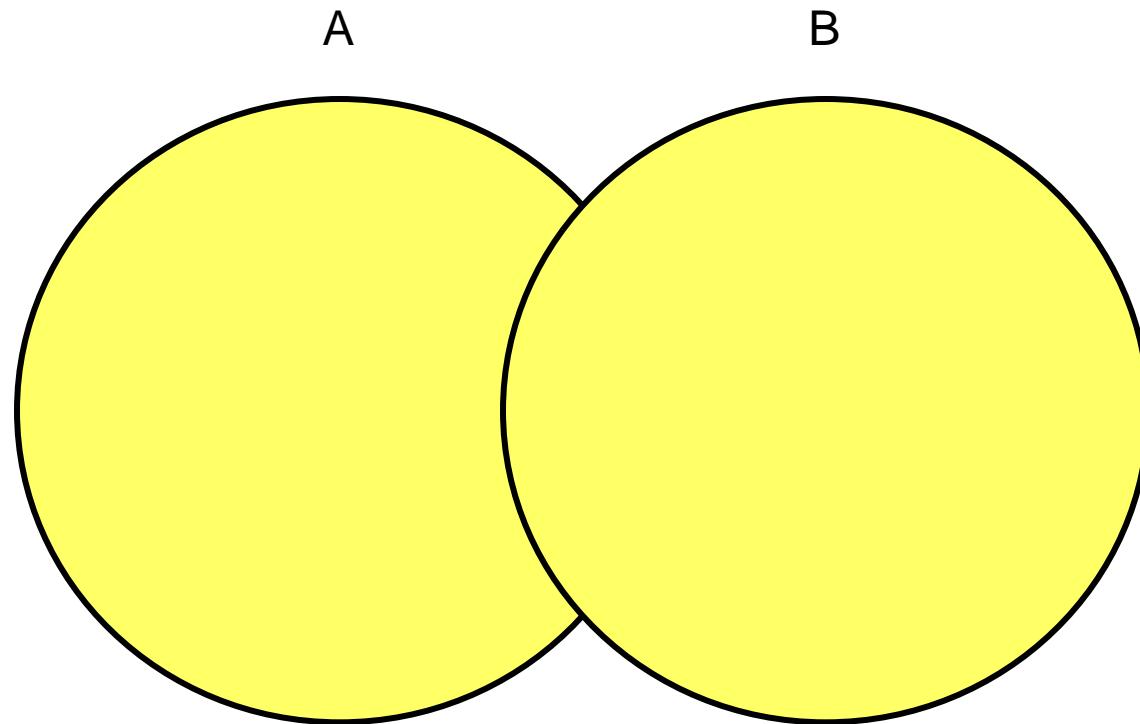
The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees
- RETIRED_EMPLOYEES: Provides details regarding all past employees

Lesson Agenda

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

UNION Operator



The UNION operator returns rows from both queries after eliminating duplications.

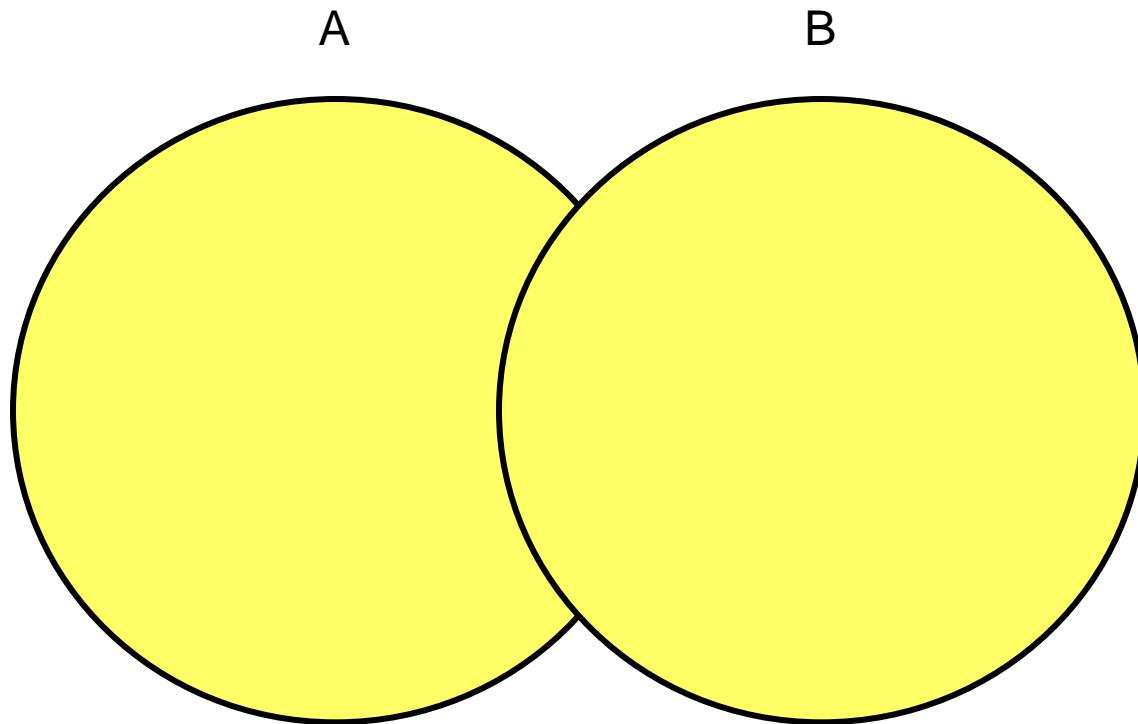
Using the UNION Operator

Display the job details of all the current and retired employees.
Display each job only once.

```
SELECT job_id  
FROM   employees  
UNION  
SELECT job_id  
FROM   retired_employees
```

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

UNION ALL Operator



The UNION ALL operator returns rows from both queries, including all duplications.

Using the UNION ALL Operator

Display the jobs and departments of all current and previous employees.

```
SELECT job_id, department_id  
FROM employees  
UNION ALL  
SELECT job_id, department_id  
FROM retired_employees  
ORDER BY job_id;
```

JOB_ID	DEPARTMENT_ID
1 AC_ACCOUNT	110
2 AC_MGR	110
3 AD_ASST	10
4 AD_PRES	90
5 AD_PRES	90
6 AD_VP	90
7 AD_VP	80
8 AD_VP	90
9 AD_VP	90

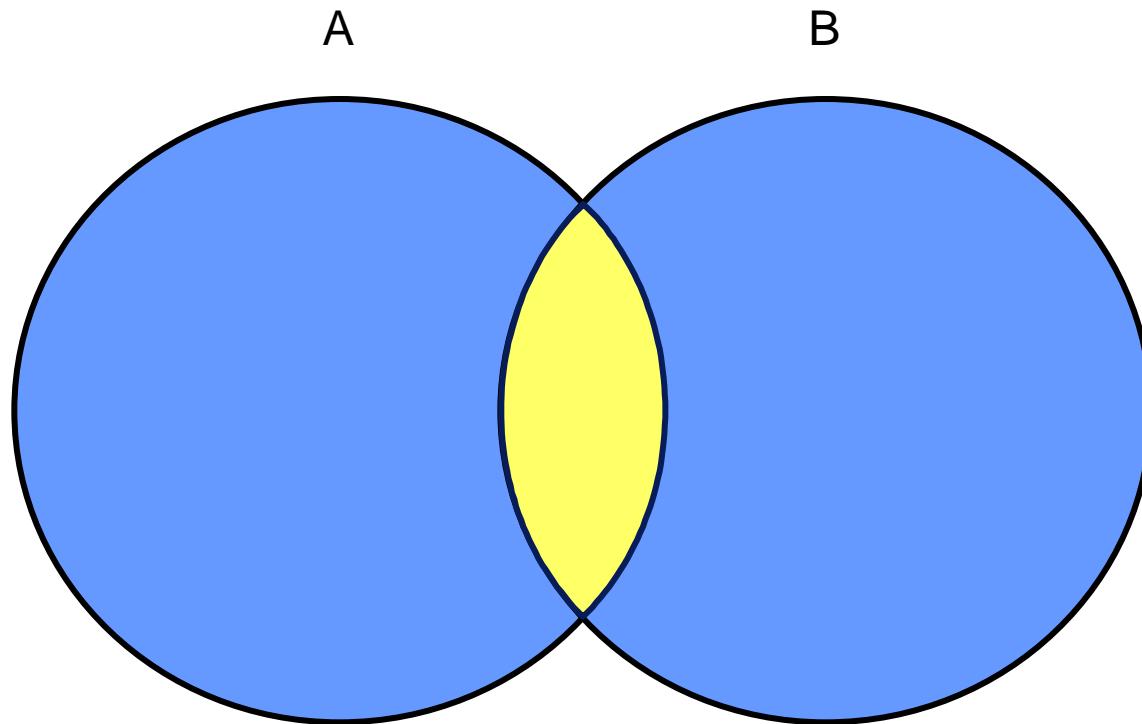
...

28	SA_REP	80
29	SA_REP	80
30	SA_REP	(null)
31	ST_CLERK	50
32	ST_CLERK	50
33	ST_CLERK	50
34	ST_CLERK	50
35	ST_MAN	50

Lesson Agenda

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

INTERSECT Operator



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

Using the INTERSECT Operator

Display the common manager IDs and department IDs of current and previous employees.

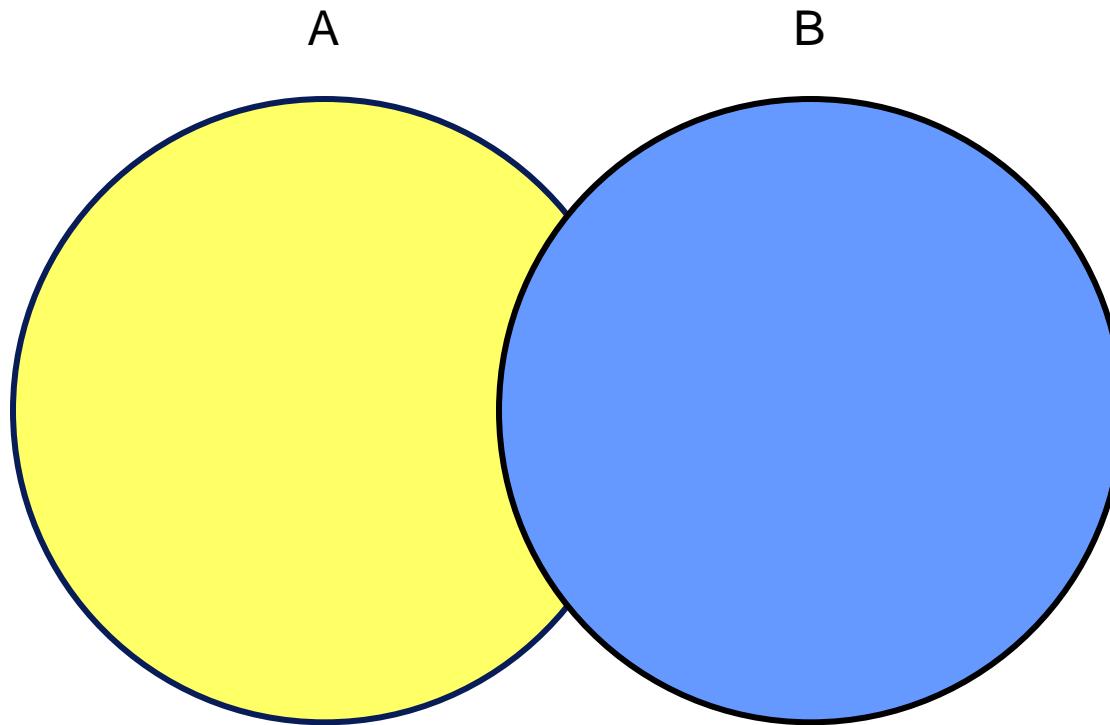
```
SELECT manager_id,department_id  
FROM employees  
INTERSECT  
SELECT manager_id,department_id  
FROM retired_employees
```

	MANAGER_ID	DEPARTMENT_ID
1	149	80

Lesson Agenda

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

MINUS Operator



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

Using the MINUS Operator

Display the employee IDs and Job IDs of those employees who works in the sales department.

```
SELECT employee_id, job_id  
FROM employees  
WHERE department_id = 80  
MINUS  
SELECT employee_id, job_id  
FROM retired_employees  
WHERE department_id = 90;
```

	EMPLOYEE_ID	JOB_ID
1	149	SA_MAN
2	174	SA_REP
3	176	SA_REP

Lesson Agenda

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

Matching SELECT Statements

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

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

Matching the SELECT Statement: Example

Using the UNION operator, display the employee name, department_id, and location_id of all employees.

```
SELECT FIRST_NAME, JOB_ID, TO_DATE(hire_date) "HIRE_DATE"
FROM employees
UNION
SELECT FIRST_NAME, JOB_ID, TO_DATE(NULL) "HIRE_DATE"
FROM retired_employees;
```

	FIRST_NAME	JOB_ID	HIRE_DATE
1	Alex	PU_CLERK	(null)
2	Alexander	IT_PROG	03-JAN-06
3	Alexandera	IT_PROG	(null)
4	Bruce	IT_PROG	21-MAY-07
5	Bruk	IT_PROG	(null)
6	Curtis	ST_CLERK	29-JAN-05
7	Dany	FI_ACCOUNT	(null)
8	De1	PU_MAN	(null)
9	Diana	IT_PROG	07-FEB-07

...

ORACLE®

Lesson Agenda

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

Using the ORDER BY Clause in Set Operations

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

Quiz

Identify two set operator guidelines.

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

Summary

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

- UNION to return all distinct rows
- UNION ALL to return all rows, including duplicates
- INTERSECT to return all rows that are shared by both queries
- MINUS to return all distinct rows that are selected by the first query, but not by the second
- ORDER BY only at the very end of the statement

Practice 9: Overview

In this practice, you create reports by using:

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

10

Managing Tables Using DML Statements

Objectives

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

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

Lesson Agenda

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

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.

Adding a New Row to 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

70 Public Relations

100

1700

New
row

Insert new row
into the
DEPARTMENTS table.



	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	70	Public Relations	100	1700
2	10	Administration	200	1700
3	20	Marketing	201	1800
4	50	Shipping	124	1500
5	60	IT	103	1400
6	80	Sales	149	2500
7	90	Executive	100	1700
8	110	Accounting	205	1700
9	190	Contracting	(null)	1700

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.

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.

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
```

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

1 rows inserted

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, 2003' , 'MON DD, YYYY' ) ,  
                'SA_REP' , 11000, 0.2, 100, 60 );
```

1 rows inserted

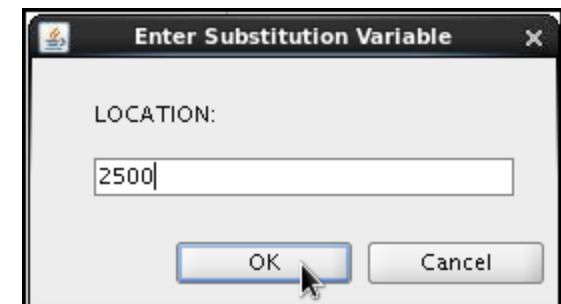
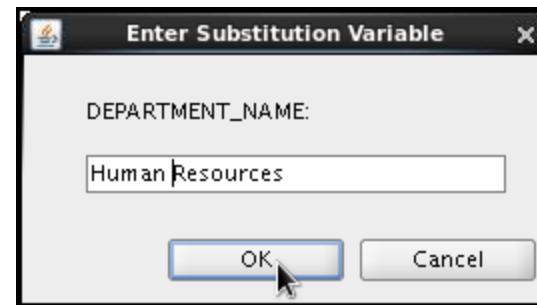
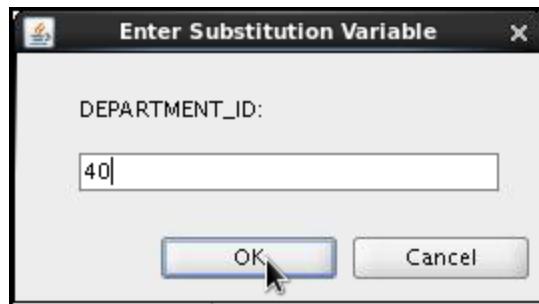
- Verify your addition.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID
1	114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-03	SA_REP	11000	0.2	100

Creating a Script

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

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



Copying Rows from Another Table

- Write your INSERT statement with a subquery:

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

5 rows inserted.

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

Lesson Agenda

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

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

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).

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.



Updating Two Columns with a Subquery

Update employee 103's job and salary to match those of employee 205.

```
UPDATE      employees
SET          ( job_id,salary )   =  (SELECT    job_id,salary
                                      FROM      employees
                                      WHERE     employee_id = 205 )
WHERE        employee_id       =      103;
```

```
1 rows updated
```

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



Lesson Agenda

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

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

DELETE Statement

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

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

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

Deleting Rows Based on Another Table

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

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

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;
```

Lesson Agenda

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

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

Database Transactions: Start and End

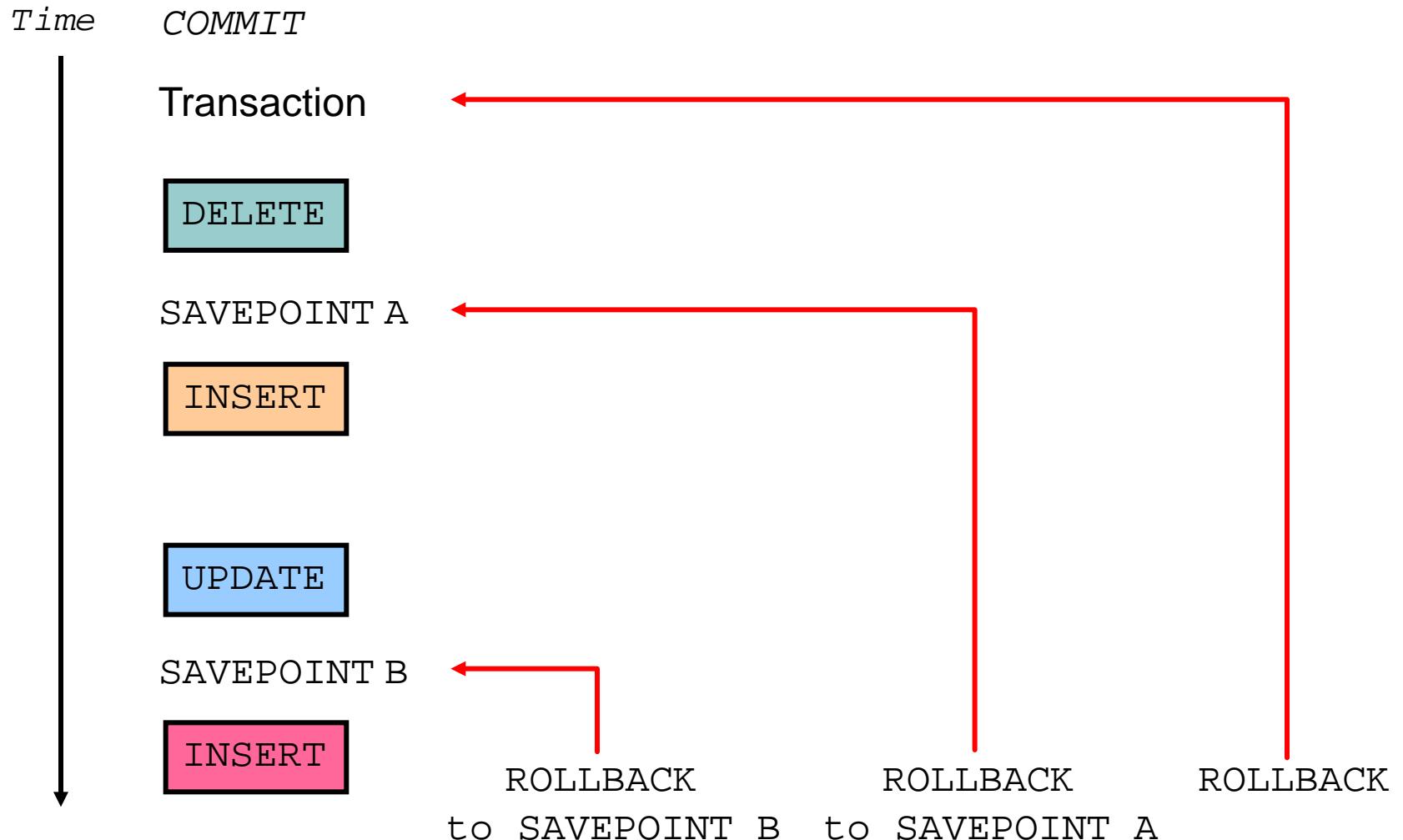
- Begin when the first DML SQL statement is executed.
- End with one of the following events:
 - A COMMIT or ROLLBACK statement is issued.
 - A DDL or DCL statement executes (automatic commit).
 - The user exits SQL Developer or SQL*Plus.
 - The system crashes.

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

Explicit Transaction Control Statements



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.
```

Implicit Transaction Processing

- An automatic commit occurs in the following circumstances:
 - A DDL statement issued
 - A DCL statement 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.

State of Data Before COMMIT or ROLLBACK

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

State of Data After COMMIT

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

Committing Data

- Make the changes:

```
DELETE FROM EMPLOYEES  
WHERE employee_id=113;  
1 rows deleted  
  
INSERT INTO departments  
VALUES (290, 'Corporate Tax', NULL, 1700);  
1 rows inserted
```

- Commit the changes:

```
COMMIT;  
  
committed.
```

State of Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

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

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

State of Data After ROLLBACK: Example

```
DELETE FROM test;  
4 rows deleted.
```

```
ROLLBACK;  
Rollback complete.
```

```
DELETE FROM test WHERE id = 100;  
1 row deleted.
```

```
SELECT * FROM test WHERE id = 100;  
No rows selected.
```

```
COMMIT;  
Commit complete.
```



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.

Lesson Agenda

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

Read Consistency

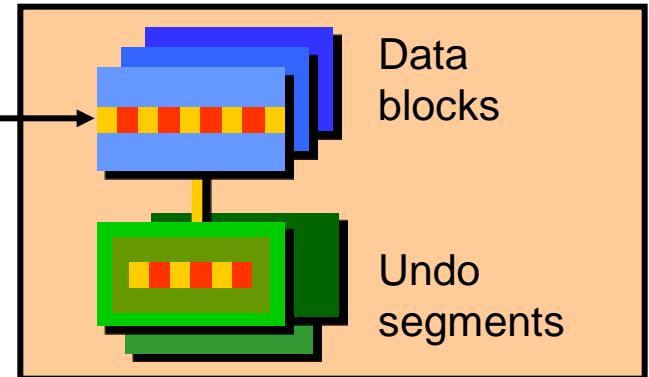
- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with the changes made by another user.
- Read consistency ensures that, on the same data:
 - Readers do not wait for writers
 - Writers do not wait for readers
 - Writers wait for writers

Implementing Read Consistency

User A



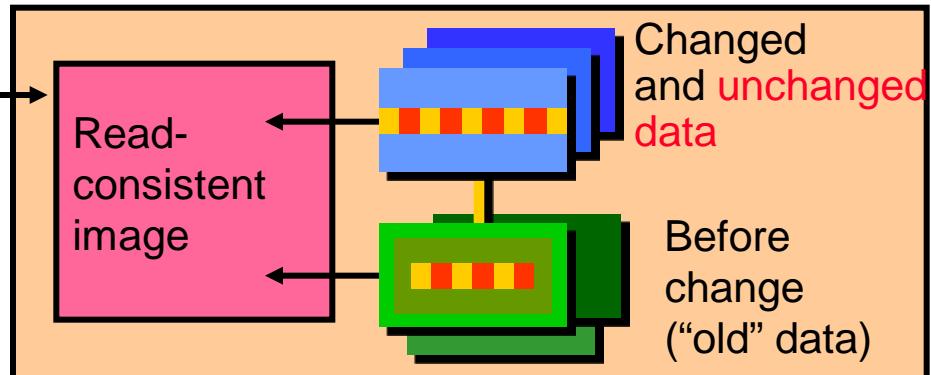
```
UPDATE employees  
SET salary = 7000  
WHERE last_name = 'Grant';
```



User B



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



Lesson Agenda

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

FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the EMPLOYEES table where job_id is SA_REP.

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

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

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.



Quiz

The following statements produce the same results:

```
DELETE FROM copy_emp;
```

```
TRUNCATE TABLE copy_emp;
```

- a. True
- b. False

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

Practice 10: Overview

This practice covers the following topics:

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

11

Introduction to Data Definition Language

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

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

Database Objects

Object	Description
Table	Is the basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative name to an object

Naming Rules

Table names and column names must:

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

Lesson Agenda

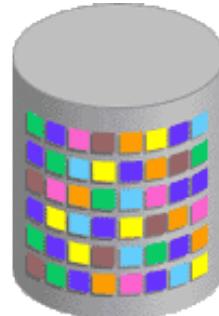
- Database objects
 - Naming rules
- **CREATE TABLE statement**
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

CREATE TABLE Statement

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

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

- You specify:
 - The table name
 - The column name, column data type, and column size



Creating Tables

- Create the table:

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

- Confirm table creation:

```
DESCRIBE dept
```

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

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

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	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE).
RAW and LONG RAW	Raw binary data
BLOB	Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table



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



DEFAULT Option

- Specify a default value for a column during the CREATE table.

```
... 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) ;
table HIRE_DATES created.
```

Lesson Agenda

- Database objects
 - Naming rules
- CREATE TABLE statement
- Data types
- **Overview of constraints:** NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

Including Constraints

- Constraints enforce rules at the table level.
- Constraints ensure the consistency and integrity of the database.
- The following constraint types are valid:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK



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.

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, ...),
```

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

NOT NULL Constraint

Ensures that null values are not permitted for the column:

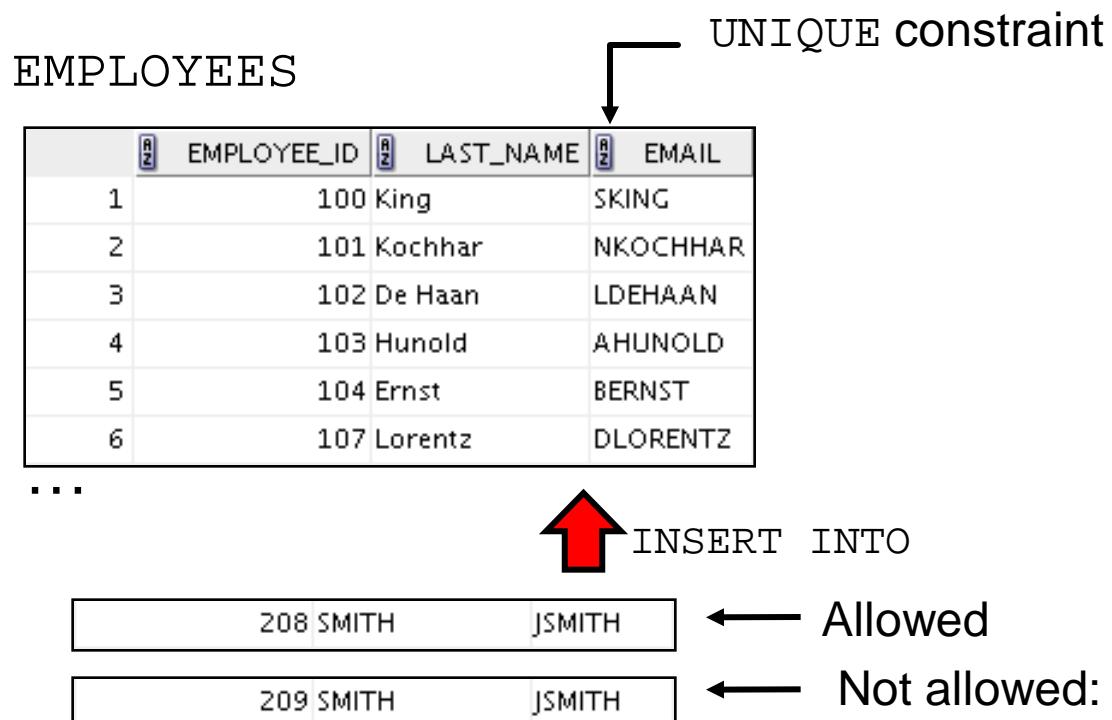
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander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

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

↑
NOT NULL
constraint

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

UNIQUE Constraint

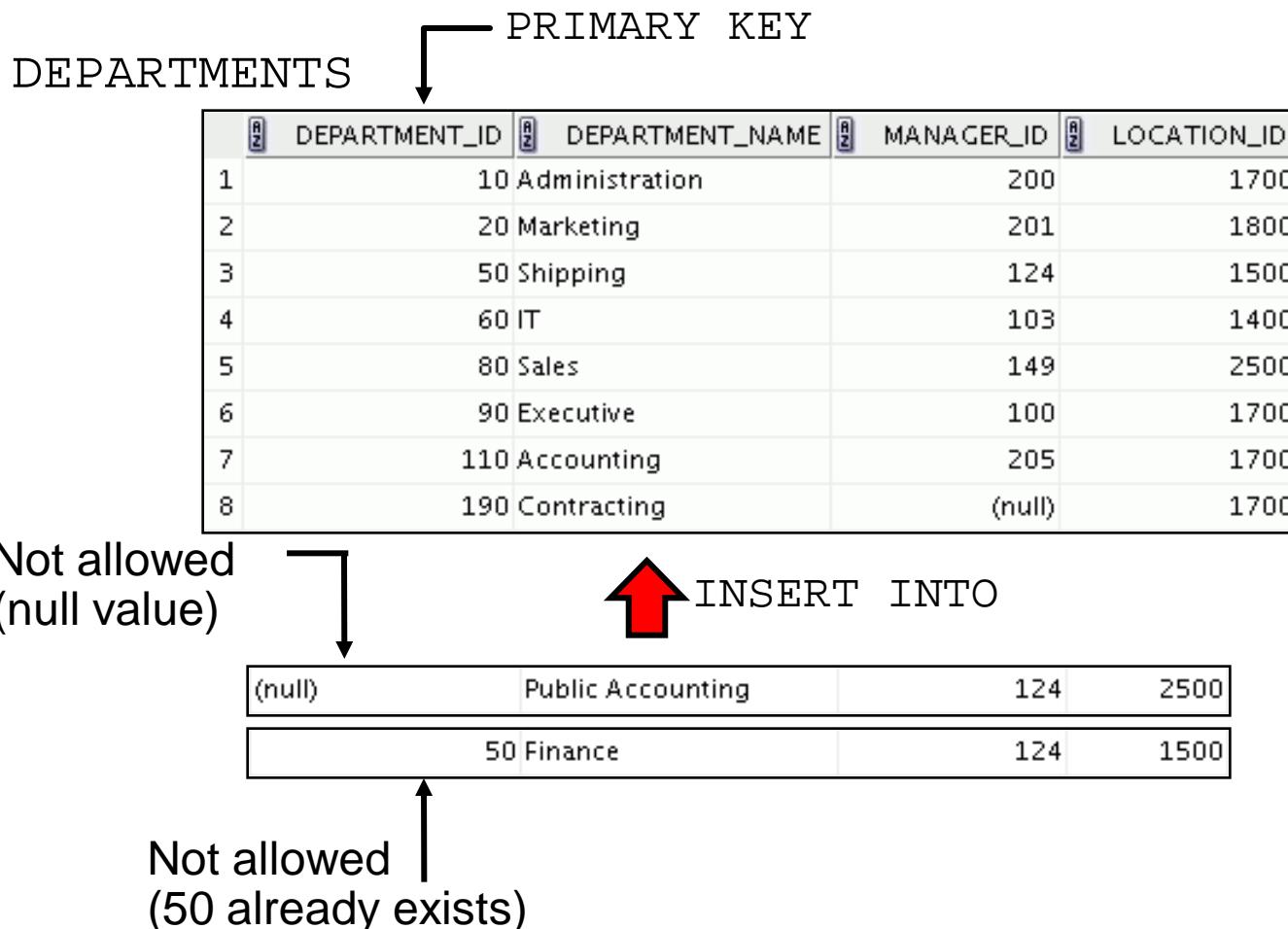


UNIQUE Constraint

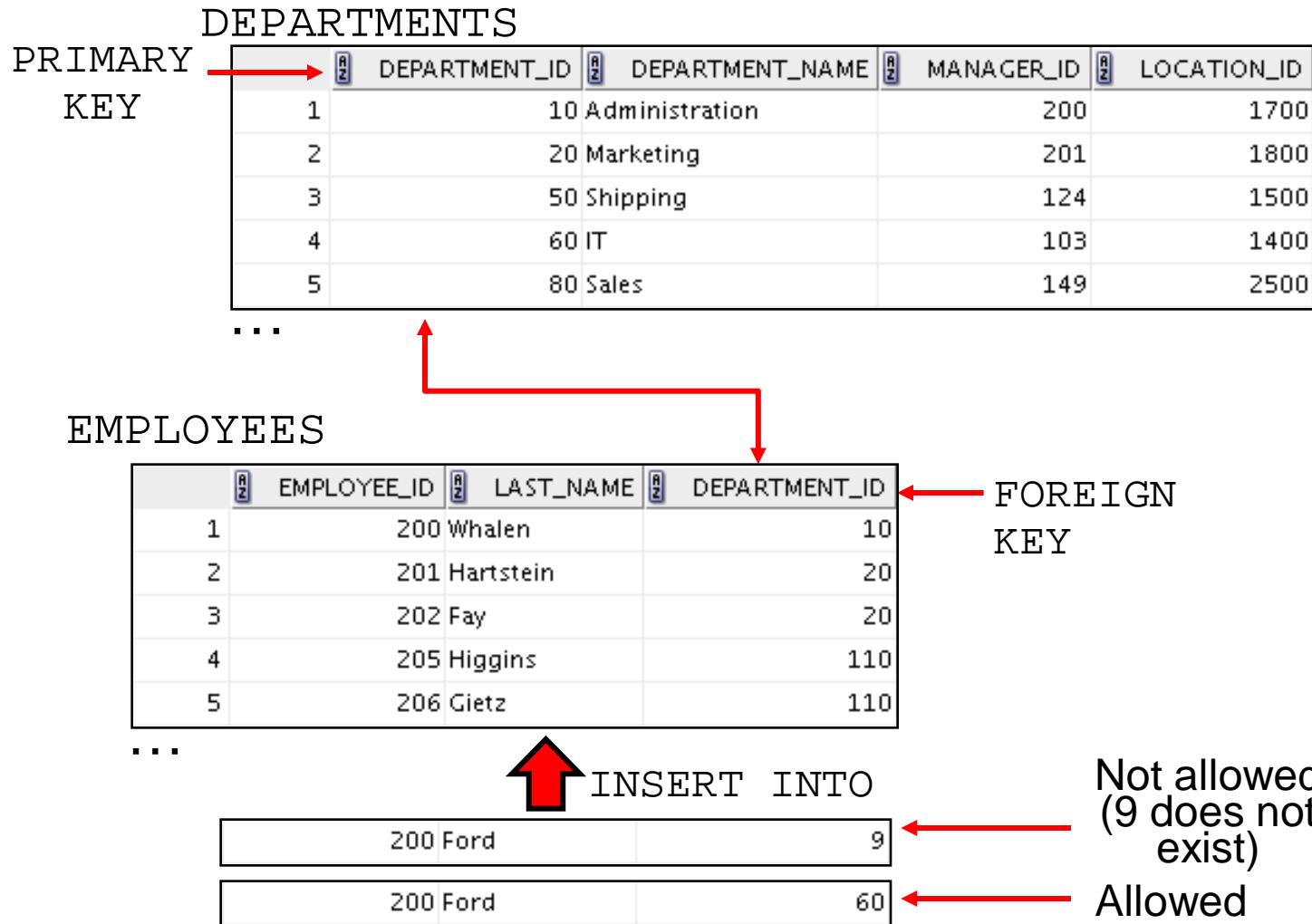
Defined at either the table level or the column level:

```
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) );
```

PRIMARY KEY Constraint



FOREIGN KEY Constraint



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));
```

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

CHECK Constraint

- It defines a condition that each row must satisfy.
- It cannot reference columns from other tables.

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

CREATE TABLE: Example

```
CREATE TABLE teach_emp (
    empno      NUMBER(5) PRIMARY KEY,
    ename      VARCHAR2(15) NOT NULL,
    job        VARCHAR2(10),
    mgr        NUMBER(5),
    hiredate   DATE DEFAULT (sysdate),
    photo      BLOB,
    sal        NUMBER(7,2),
    deptno    NUMBER(3) NOT NULL
        CONSTRAINT admin_dept_fkey REFERENCES
            departments(department_id));
```

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 (ORA1.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.

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 (ORA1.JHIST_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.
```

Lesson Agenda

- Database objects
 - Naming rules
- Data types
- CREATE TABLE statement
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

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.

Creating a Table Using a Subquery

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

table DEPT80 created.

```
DESCRIBE dept80
```

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



Lesson Agenda

- Database objects
 - Naming rules
- Data types
- CREATE TABLE statement
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

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

ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
              [, column datatype]...);
```

```
ALTER TABLE table
MODIFY        (column datatype [DEFAULT expr]
              [, column datatype]...);
```

```
ALTER TABLE table
DROP  (column [, column] ...);
```

Adding a Column

- You use the ADD clause to add columns:

```
ALTER TABLE dept80  
ADD          ( job_id VARCHAR2(9) );
```

table DEPT80 altered.

- The new column becomes the last column:

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	149 Zlotkey		10500	29-JAN-08	(null)
2	174 Abel		11000	11-MAY-04	(null)
3	176 Taylor		8600	24-MAR-06	(null)

Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE dept80
MODIFY      (last_name VARCHAR2(30));
```

```
table DEPT80 altered.
```

- A change to the default value affects only subsequent insertions to the table.



Dropping a Column

Use the DROP COLUMN clause to drop columns that you no longer need from the table:

```
ALTER TABLE dept80
DROP (job_id);
```

```
table DEPT80 altered.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	10500	29-JAN-08
2	174	Abel	11000	11-MAY-04
3	176	Taylor	8600	24-MAR-06

SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.
- You can specify the ONLINE keyword to indicate that DML operations on the table will be allowed while marking the column or columns UNUSED.

```
ALTER TABLE <table_name>
SET UNUSED(<column_name> [ , <column_name>]);
```

OR

```
ALTER TABLE <table_name>
SET UNUSED COLUMN <column_name> [ ,<column_name>];
```

```
ALTER TABLE <table_name>
DROP UNUSED COLUMNS;
```



Read-Only Tables

You can use the ALTER TABLE syntax to:

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

```
ALTER TABLE employees READ ONLY;  
  
-- perform table maintenance and then  
-- return table back to read/write mode  
  
ALTER TABLE employees READ WRITE;
```

Lesson Agenda

- Database objects
 - Naming rules
- Data types
- CREATE TABLE statement
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE statement
- DROP TABLE statement

Dropping a Table

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

```
DROP TABLE dept80;  
table DEPT80 dropped.
```

Quiz

To do which three of the following can you use constraints?

- a. Enforce rules on the data in a table whenever a row is inserted, updated, or deleted.
- b. Prevent the dropping of a table.
- c. Prevent the creation of a table.
- d. Prevent the creation of data in a table.

Summary

In this lesson, you should have learned how to use the CREATE TABLE, ALTER TABLE, and DROP TABLE statement to create a table, modify a table and columns, and include constraints.

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

Practice 11: Overview

This practice covers the following topics:

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

Table Descriptions

B

Using SQL Developer

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

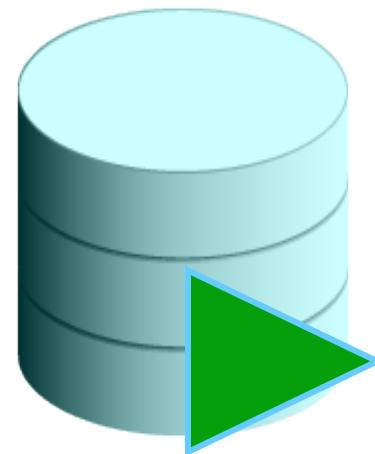
Objectives

After completing this appendix, you should be able to:

- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports
- Browse the Data Modeling options in SQL Developer

What Is Oracle SQL Developer?

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



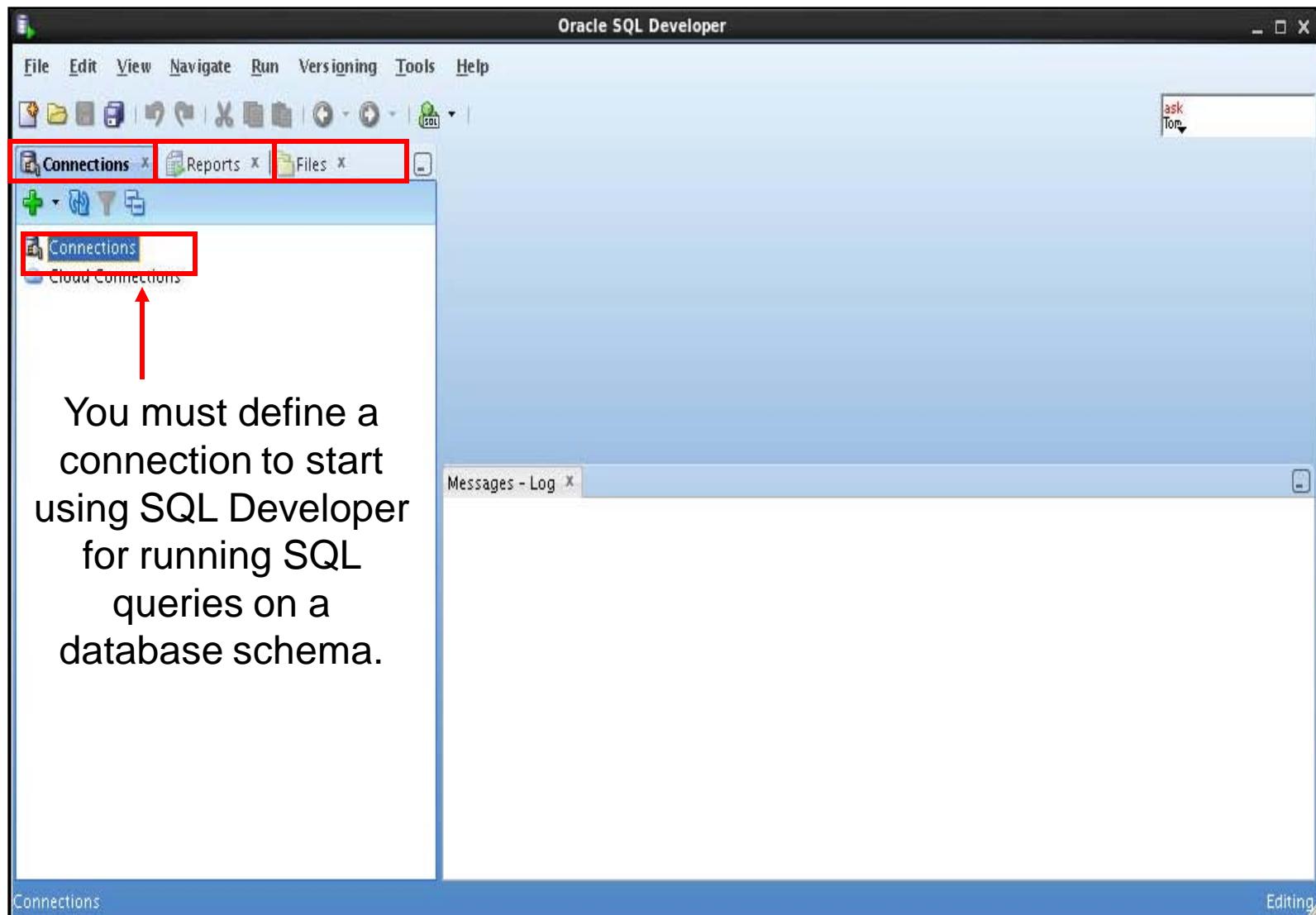
SQL Developer

ORACLE®

Specifications of SQL Developer

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

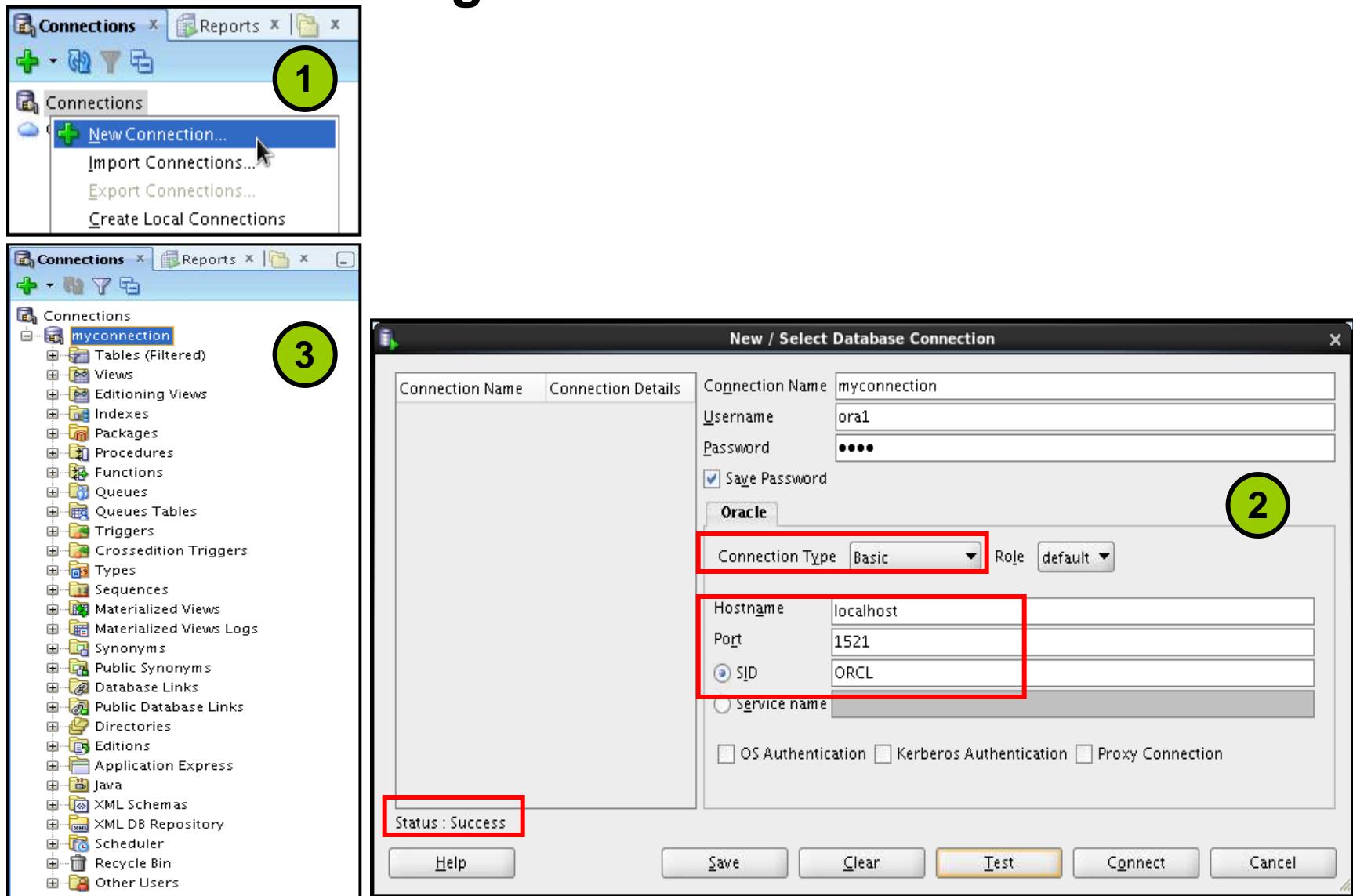
SQL Developer 3.2 Interface



Creating a Database Connection

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

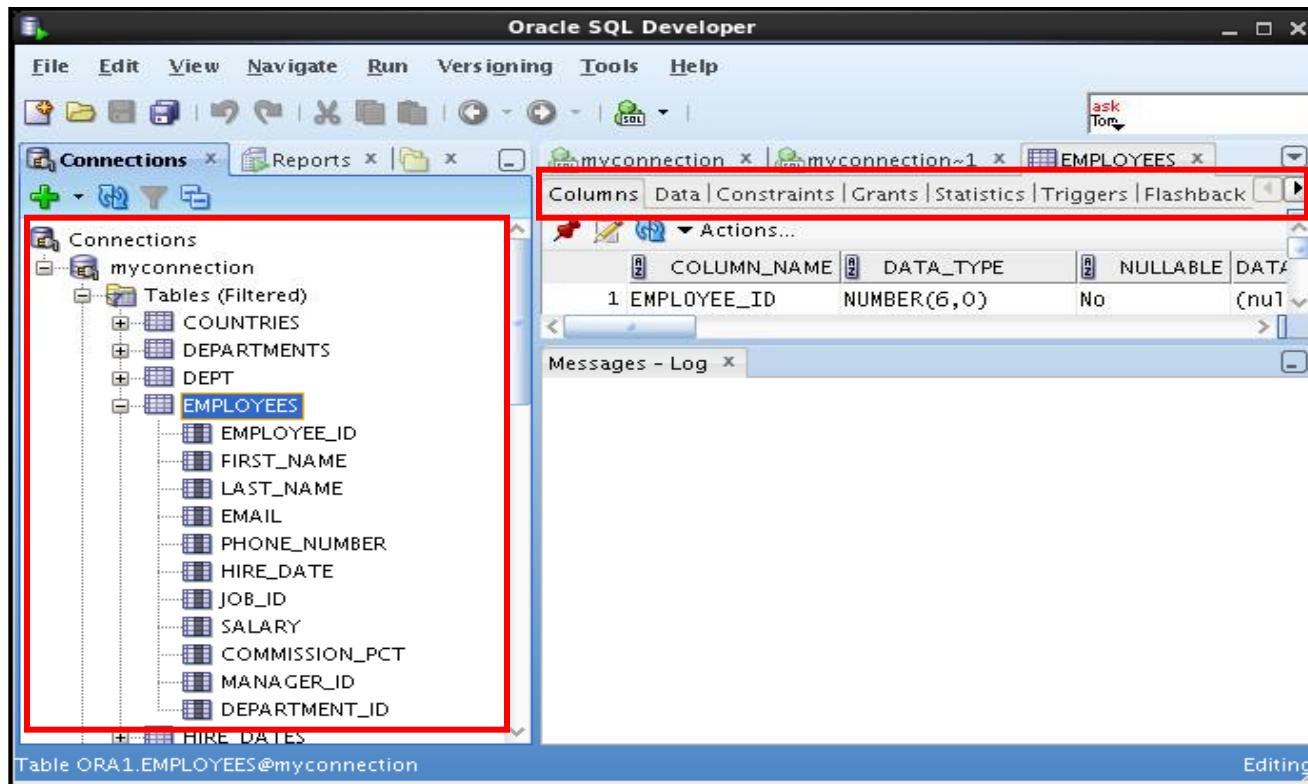
Creating a Database Connection



Browsing Database Objects

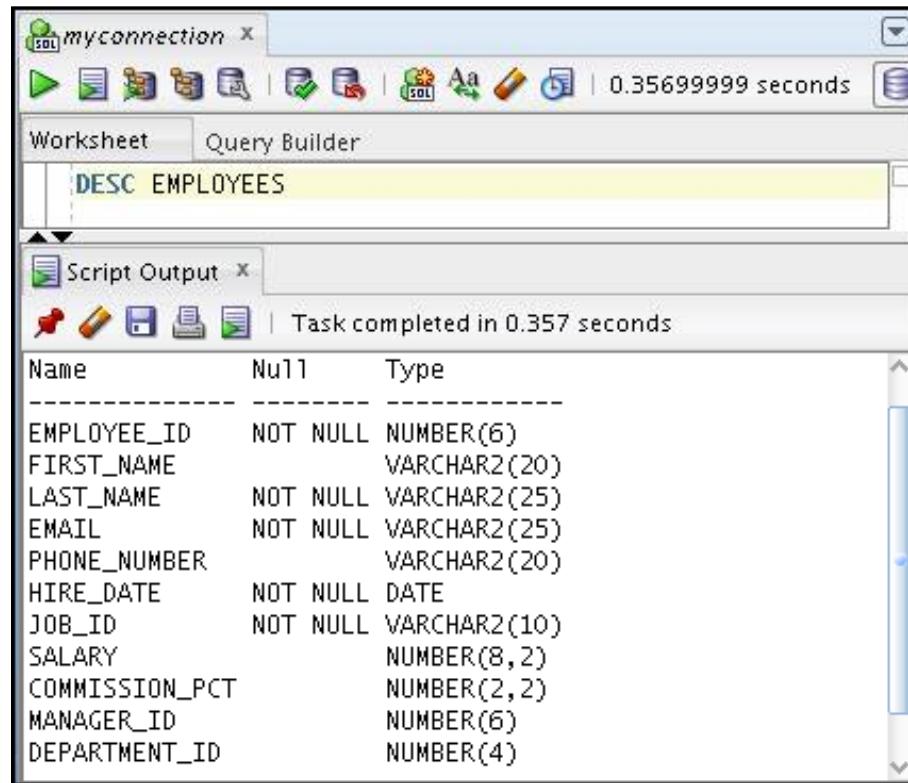
Use the Connections Navigator to:

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



Displaying the Table Structure

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

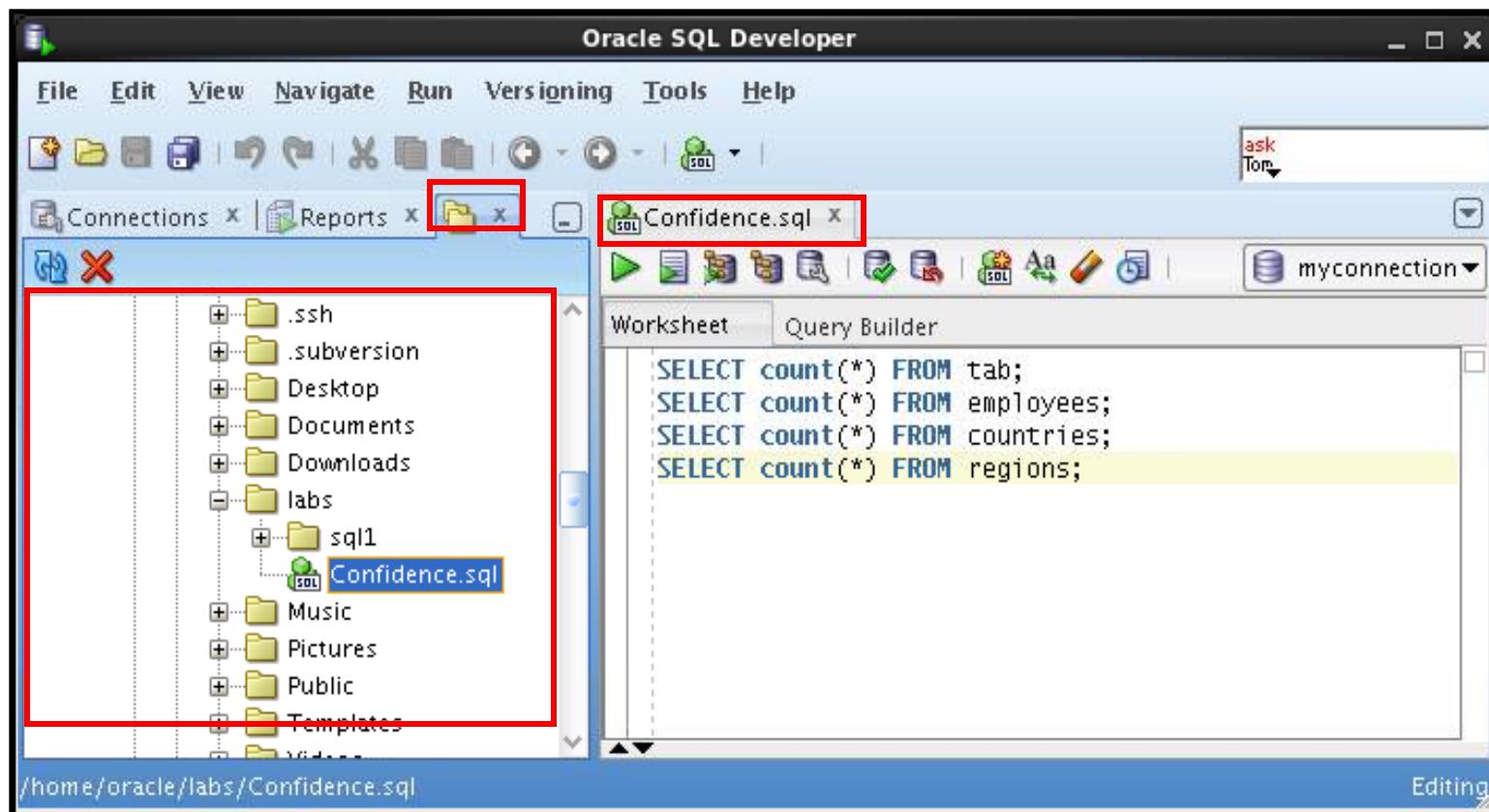


The screenshot shows the Oracle SQL Developer interface. The top window is titled "myconnection" and contains a toolbar with various icons. Below the toolbar, there are tabs for "Worksheet" and "Query Builder", with "Worksheet" selected. The main area displays the command "DESC EMPLOYEES". Below this, the "Script Output" tab is active, showing the results of the command. The results are presented as a table with columns "Name", "Null", and "Type".

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

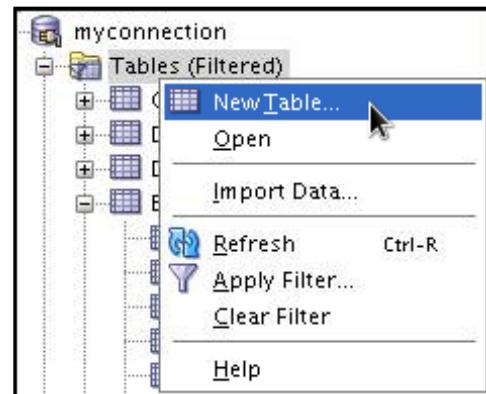
Browsing Files

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

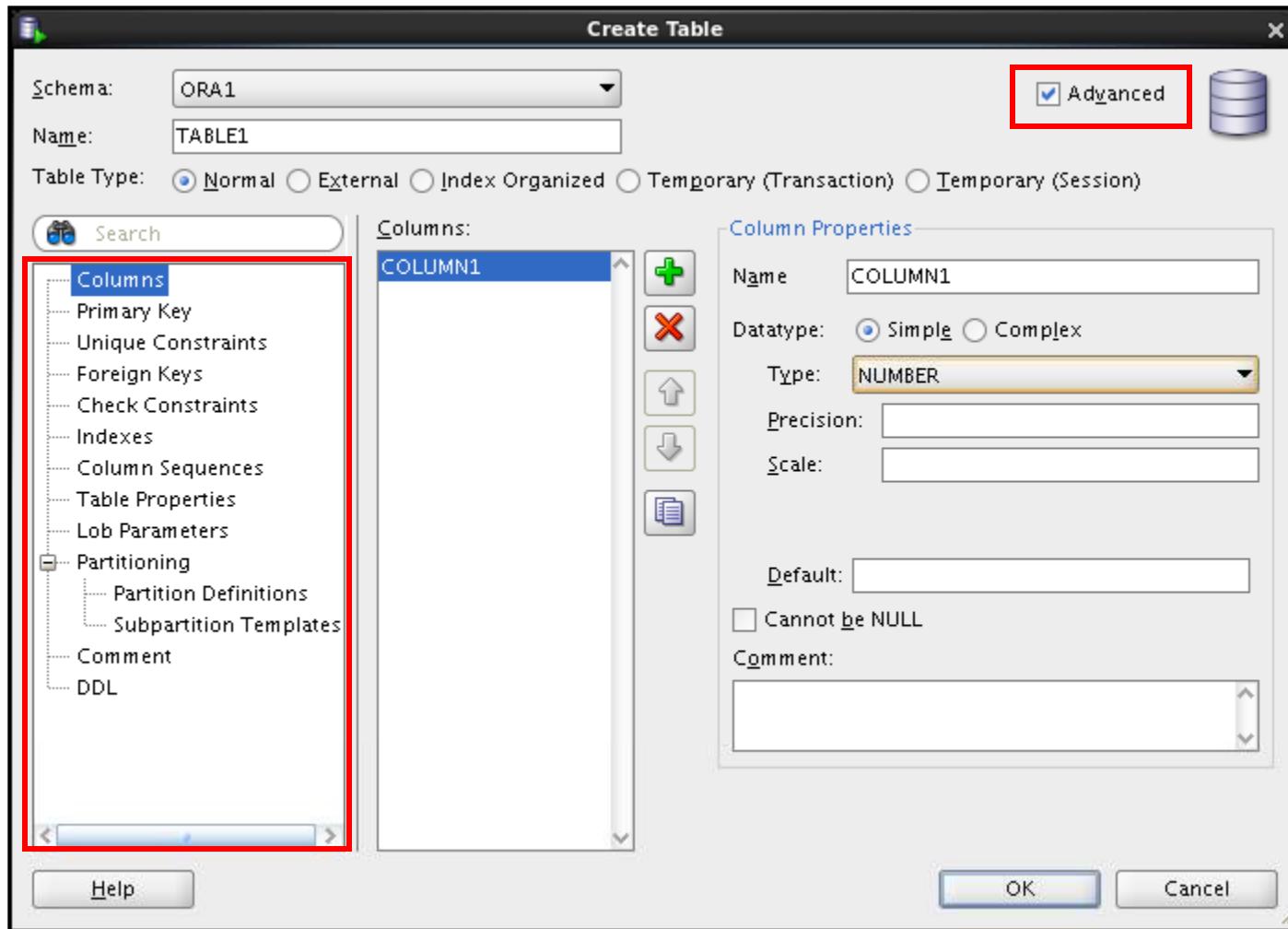


Creating a Schema Object

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

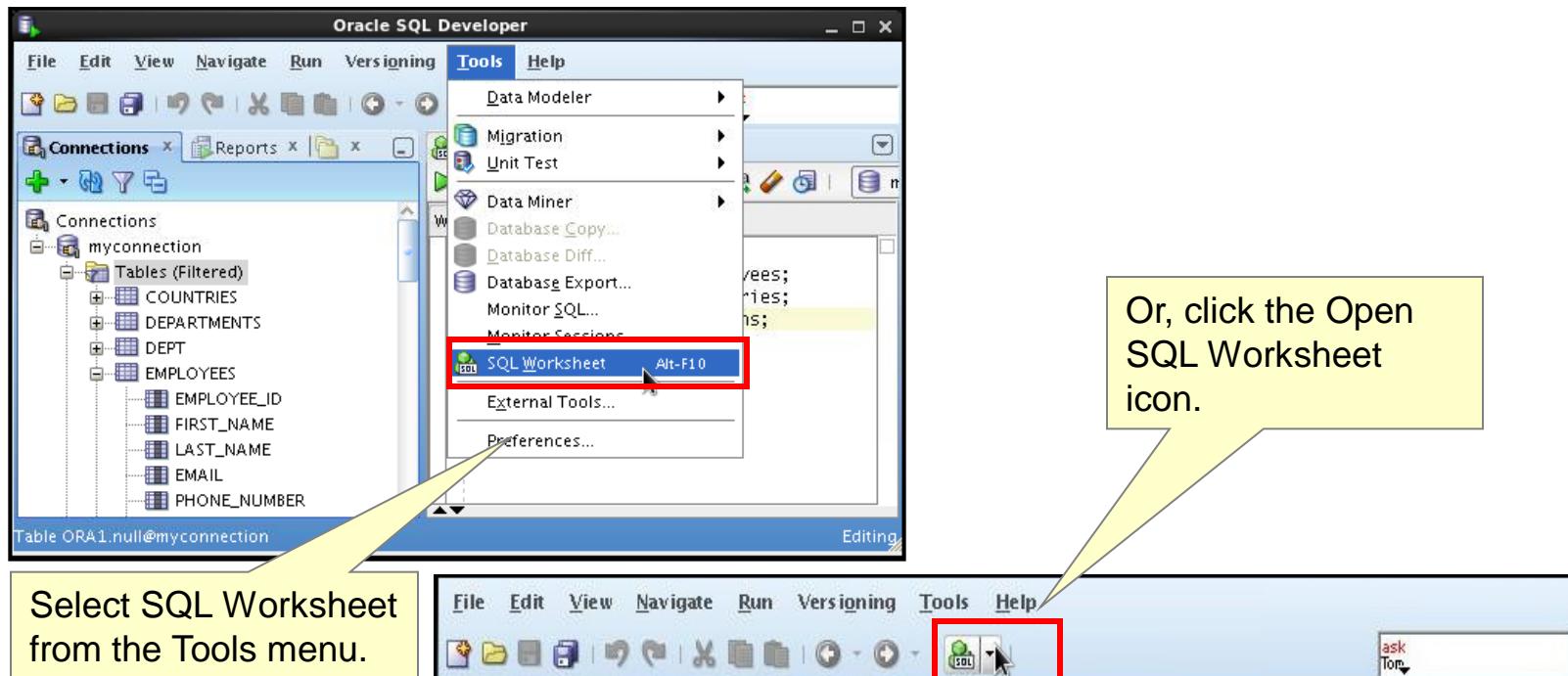


Creating a New Table: Example

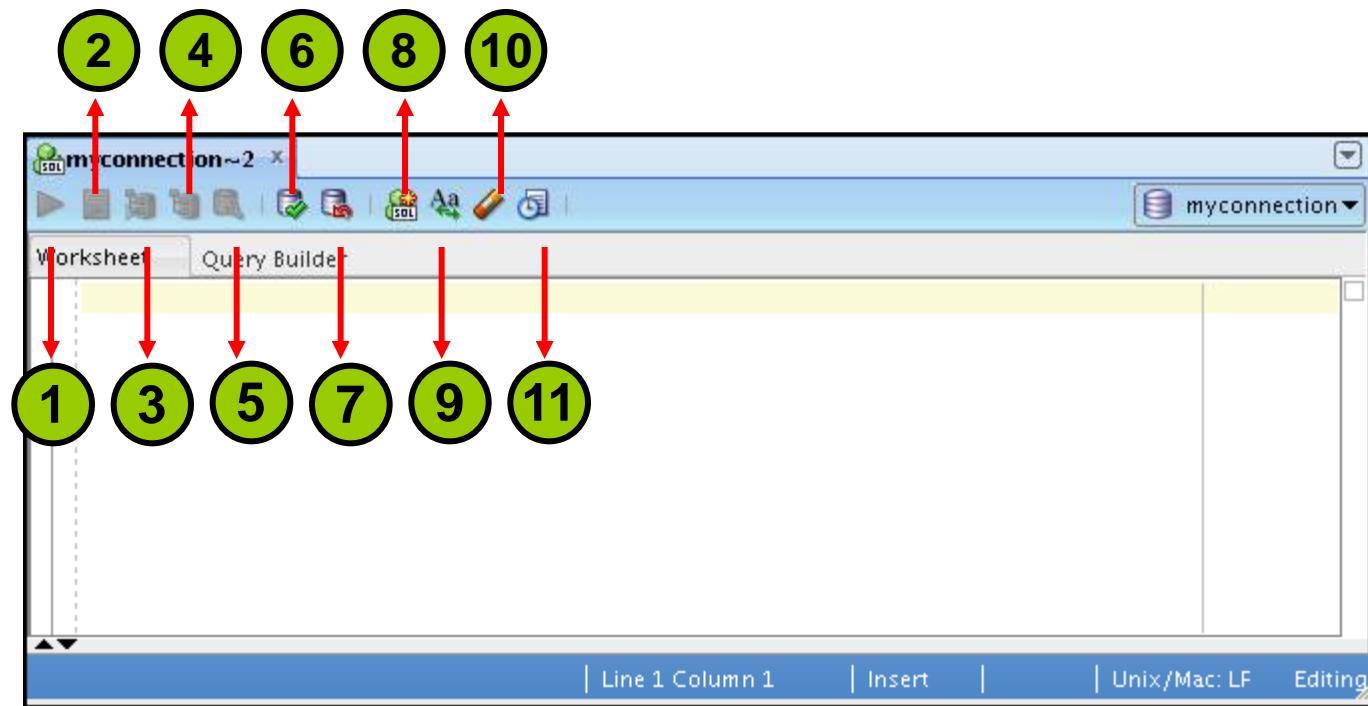


Using the SQL Worksheet

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

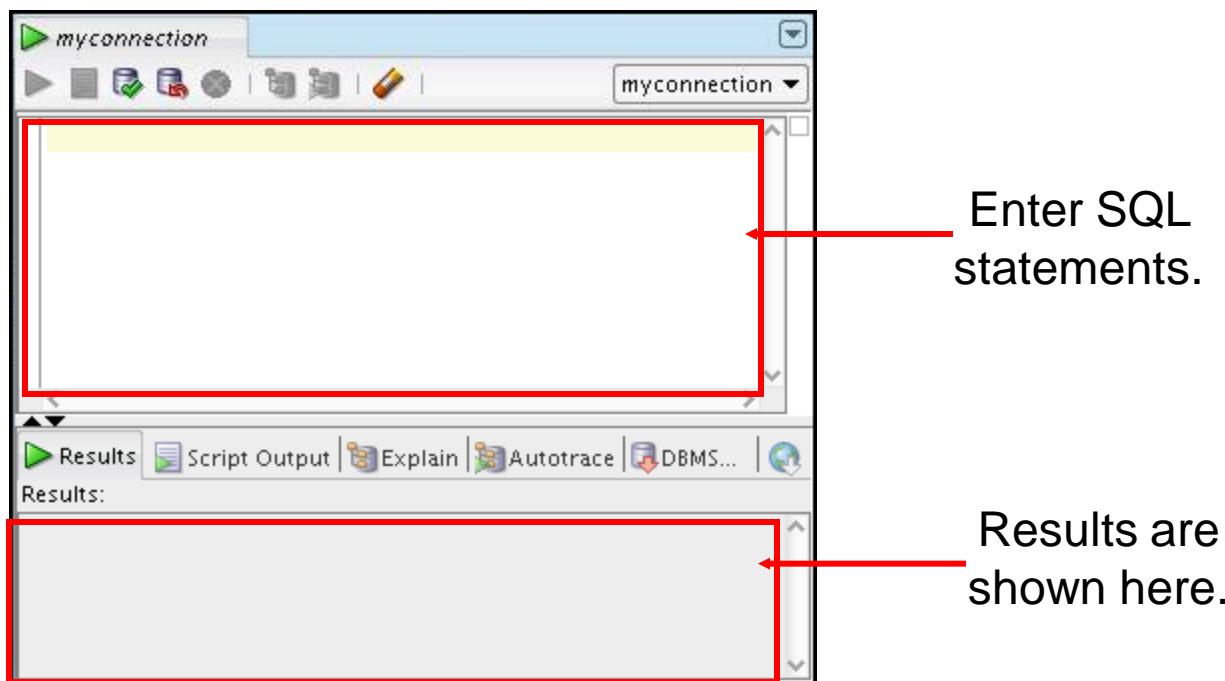


Using the SQL Worksheet



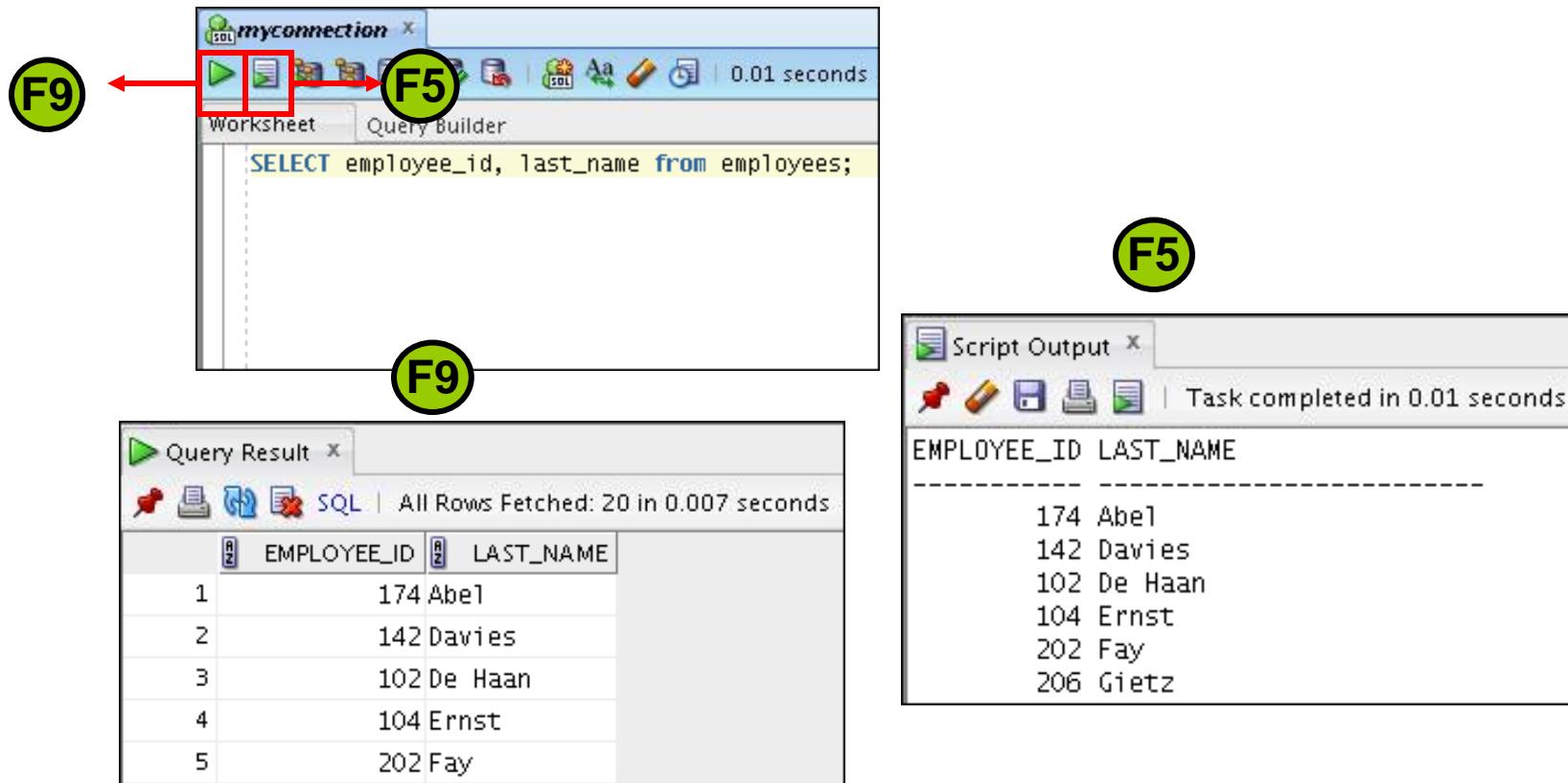
Using the SQL Worksheet

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



Executing SQL Statements

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



Saving SQL Scripts

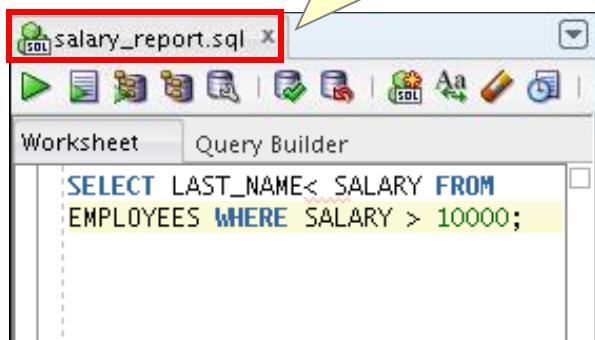
1

Click the Save icon to save your SQL statement to a file.



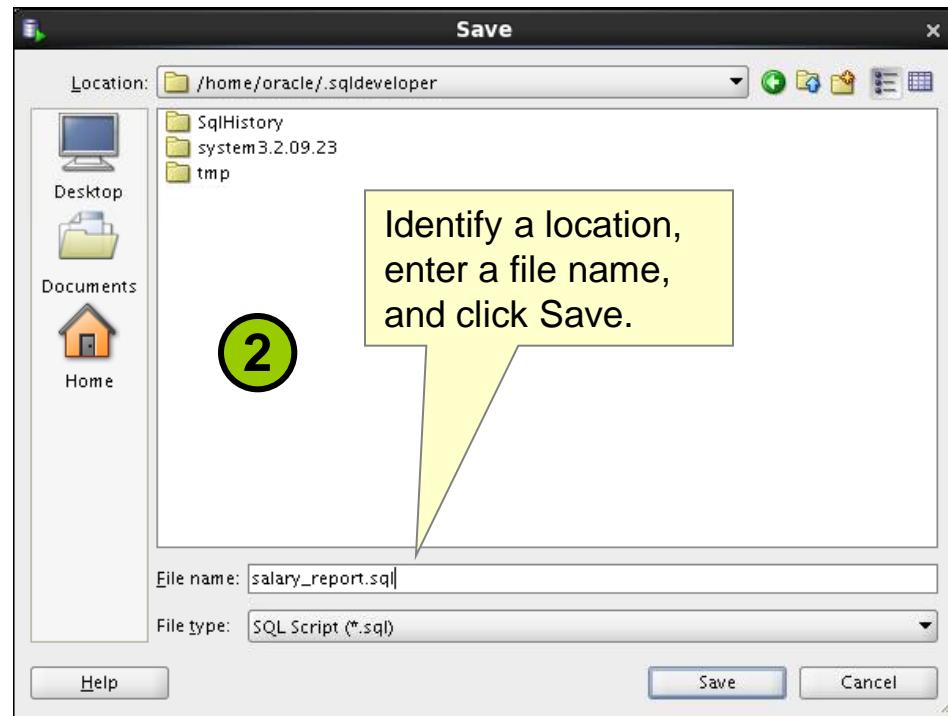
3

The contents of the saved file are visible and editable in your SQL Worksheet window.

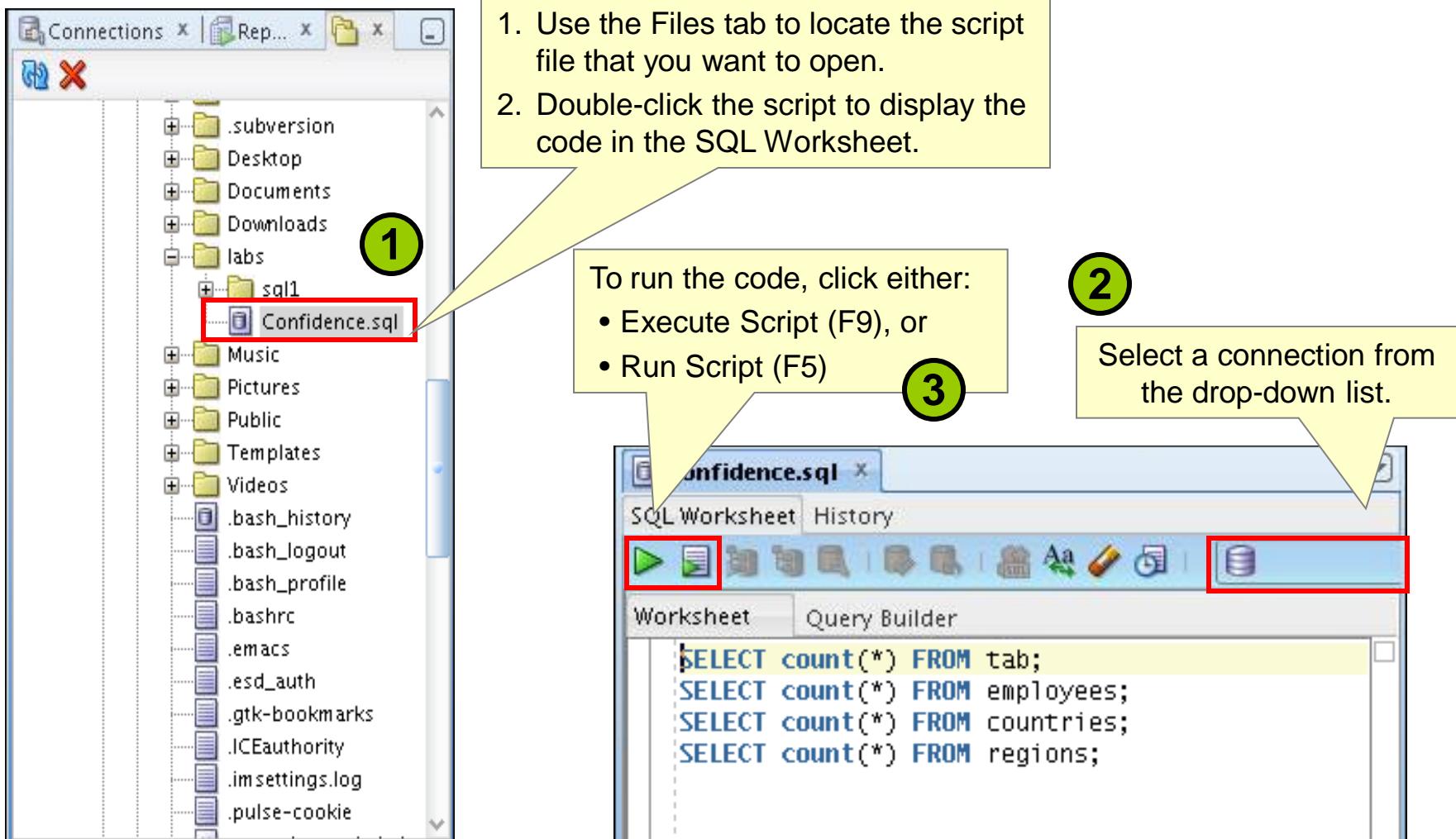


2

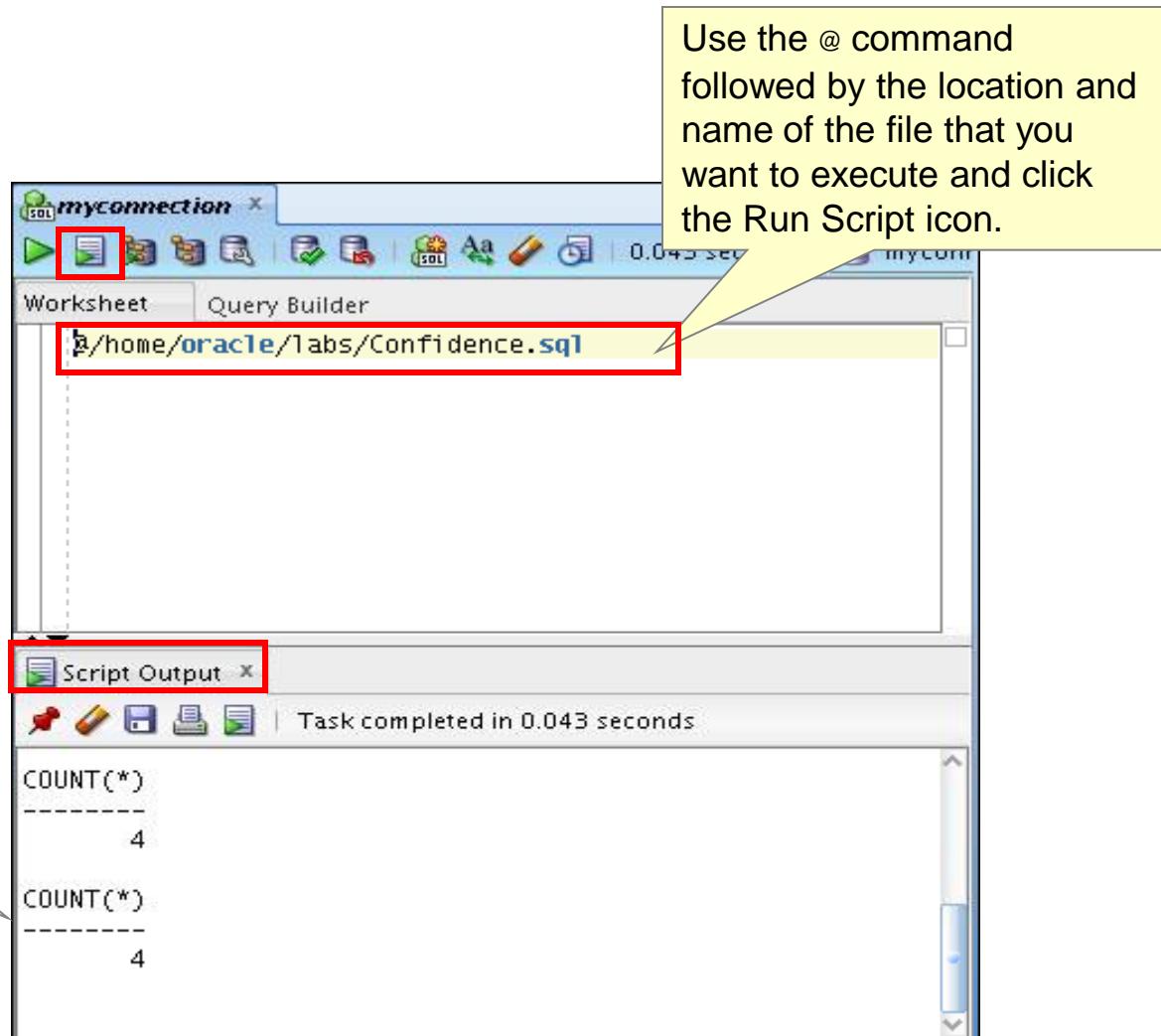
Identify a location, enter a file name, and click Save.



Executing Saved Script Files: Method 1



Executing Saved Script Files: Method 2



Formatting the SQL Code

Before formatting

The screenshot shows the Oracle SQL Developer interface. A context menu is open over a block of SQL code. The menu items include: Run Statement (F9), Run Script (F5), Create Report..., Save as Snippet..., Autotrace... (F6), Explain Plan... (F10), SQL Tuning Advisor... (Ctrl-F12), Commit (F11), Rollback (F12), To Upper/Lower/InitCap (Ctrl+Quote), Clear (Ctrl-D), SQL History (F8), Cut (Ctrl-X), Copy (Ctrl-C), Paste (Ctrl-V), Select All (Ctrl-A), Debug (Ctrl+Shift-F10), Refactoring (with a submenu), Format (Ctrl-F7) which is highlighted with a red box, Advanced Format... (Ctrl+Shift-F7), Code Template, Popup Describe, and Open Declaration (Shift-F4). The SQL code in the editor is:

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

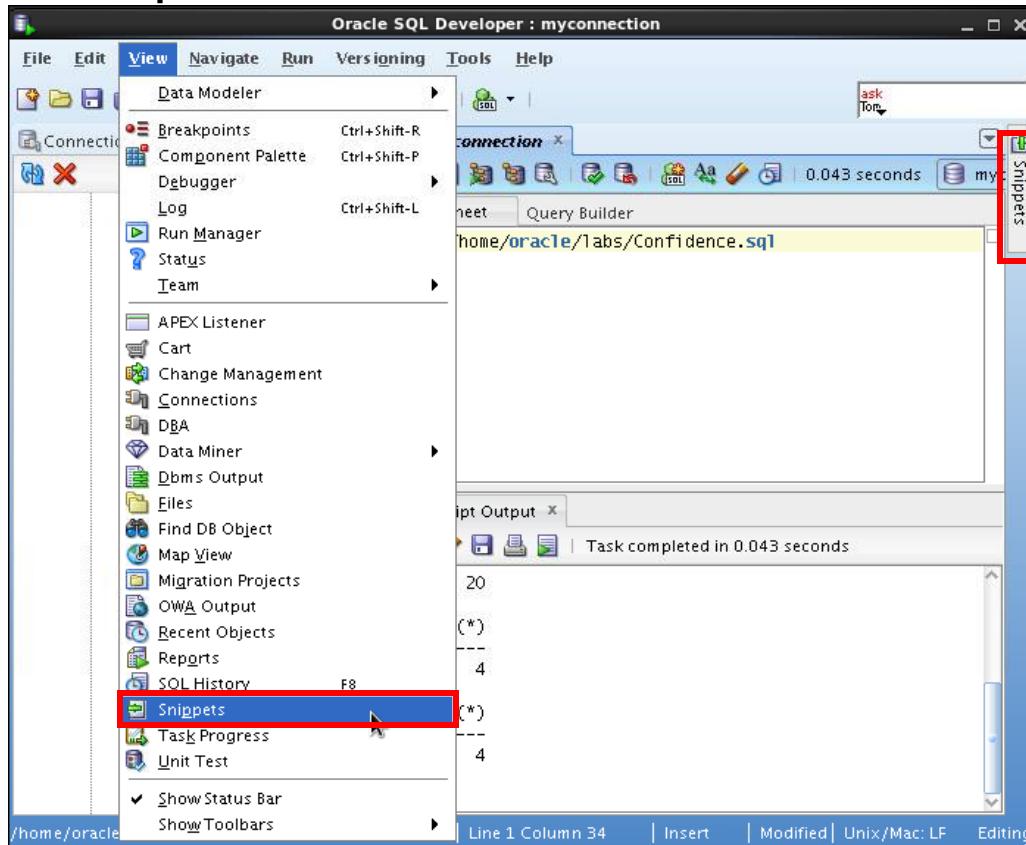
After formatting

The screenshot shows the Oracle SQL Developer interface after the SQL code has been formatted. The code is now properly indented and structured:

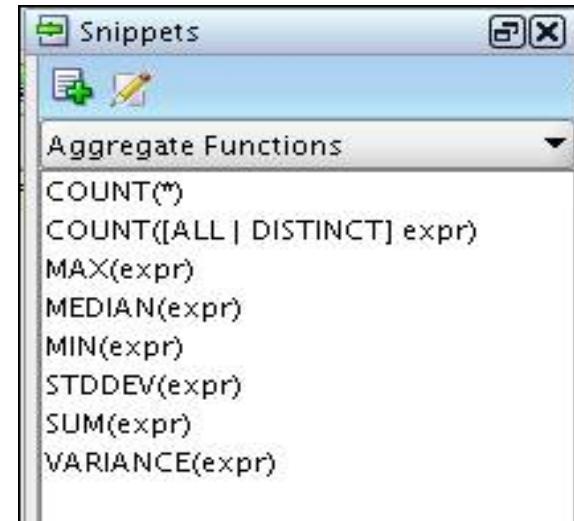
```
SELECT employee_id,  
       first_name,  
       salary  
  FROM employees e,  
        departments d  
 WHERE e.department_id=d.department_id  
   AND e.salary>3000;
```

Using Snippets

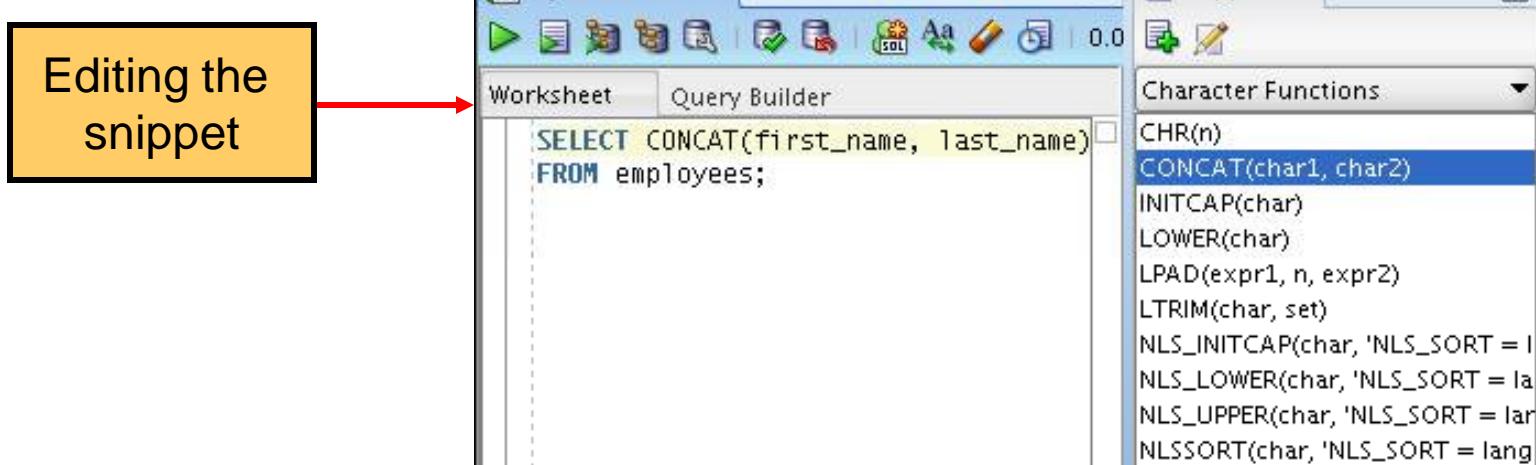
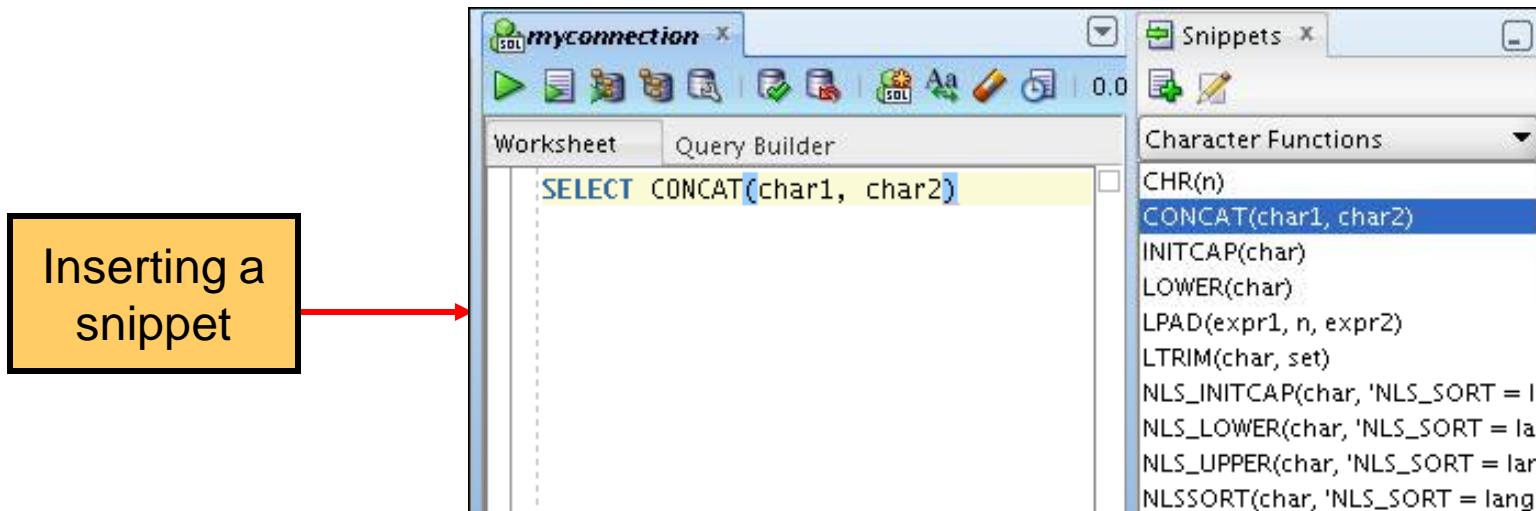
Snippets are code fragments that may be just syntax or examples.



When you place your cursor here, it shows the Snippets window. From the drop-down list, you can select the functions category that you want.

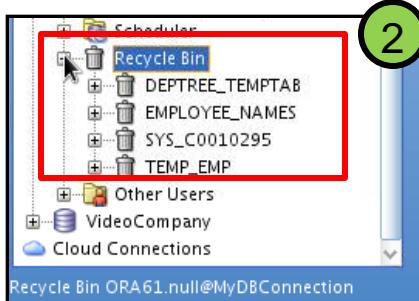
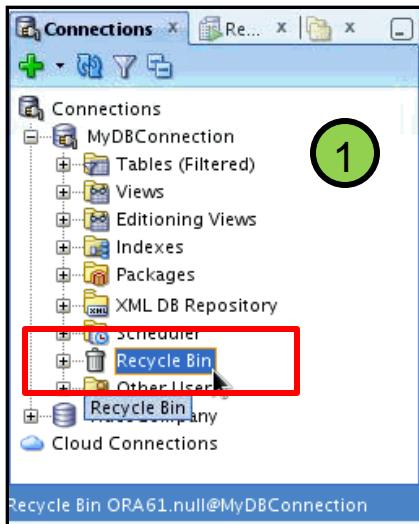


Using Snippets: Example



Using the Recycle Bin

The recycle bin holds objects that have been dropped.



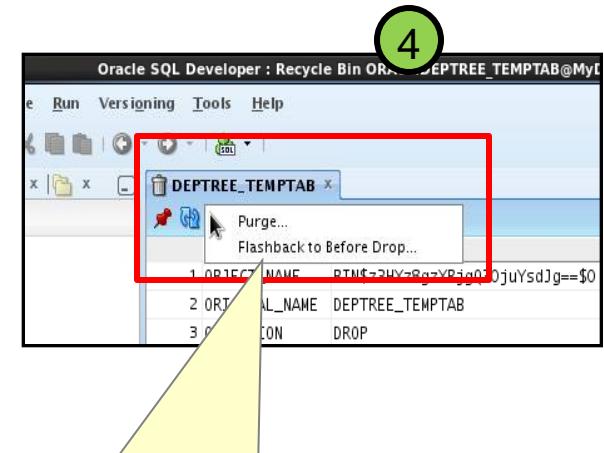
Select the operations from the drop-down Actions list.

3

DEPTREE_TEMPTAB

Actions... ▾

Name	Value
1 OBJECT_NAME	BIN\$z3HXz8gvXrgjgQ30juYsdJg==\$0
2 ORIGINAL_NAME	DEPTREE_TEMPTAB
3 OPERATION	DROP
4 TYPE	TABLE
5 TS_NAME	USERS
6 CREATETIME	2012-11-30:01:14:37
7 DROPTIME	2012-11-30:01:17:57
8 DROPSCN	3398921
9 PARTITION_NAME	(null)
10 CAN_UNDROP	YES
11 CAN_PURGE	YES
12 RELATED	94960
13 BASE_OBJECT	94960
14 PURGE_OBJECT	94960
15 SPACE	0

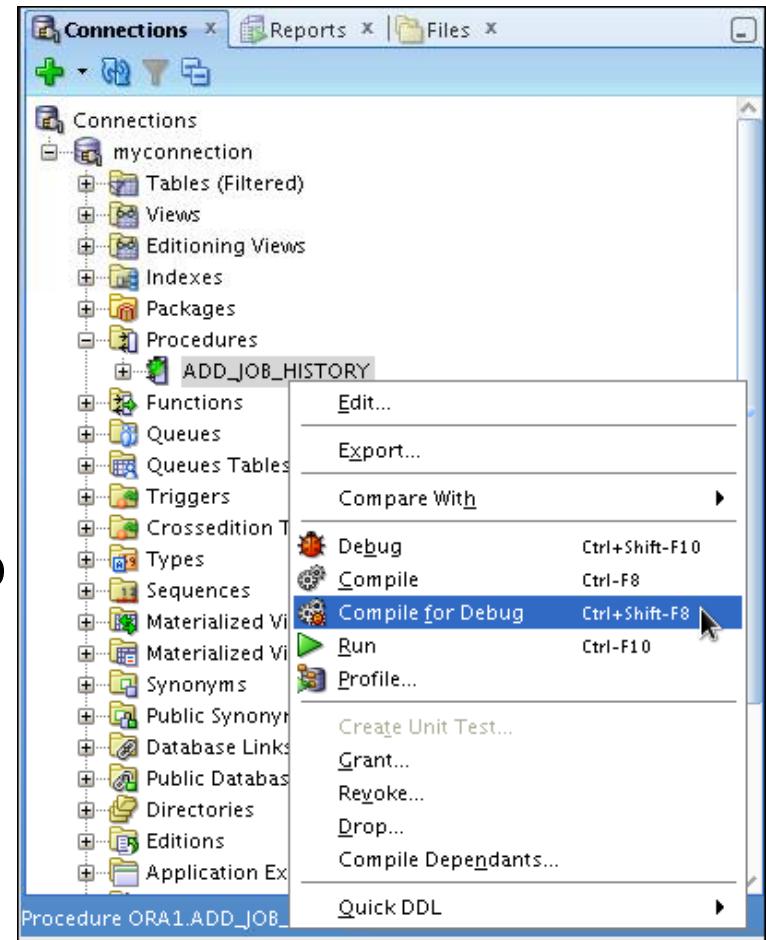


Purge: Removes the object from the Recycle bin and deletes it

Flashback to Before Drop:
Moves the object from the recycle bin back to its appropriate place in the Connections navigator display

Debugging Procedures and Functions

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



Database Reporting

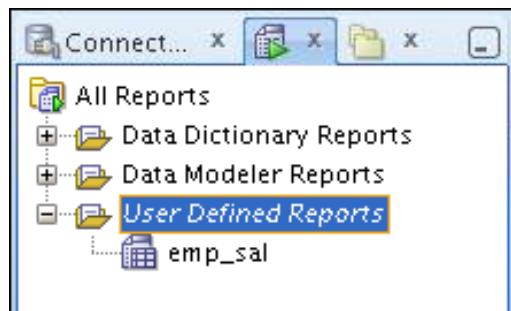
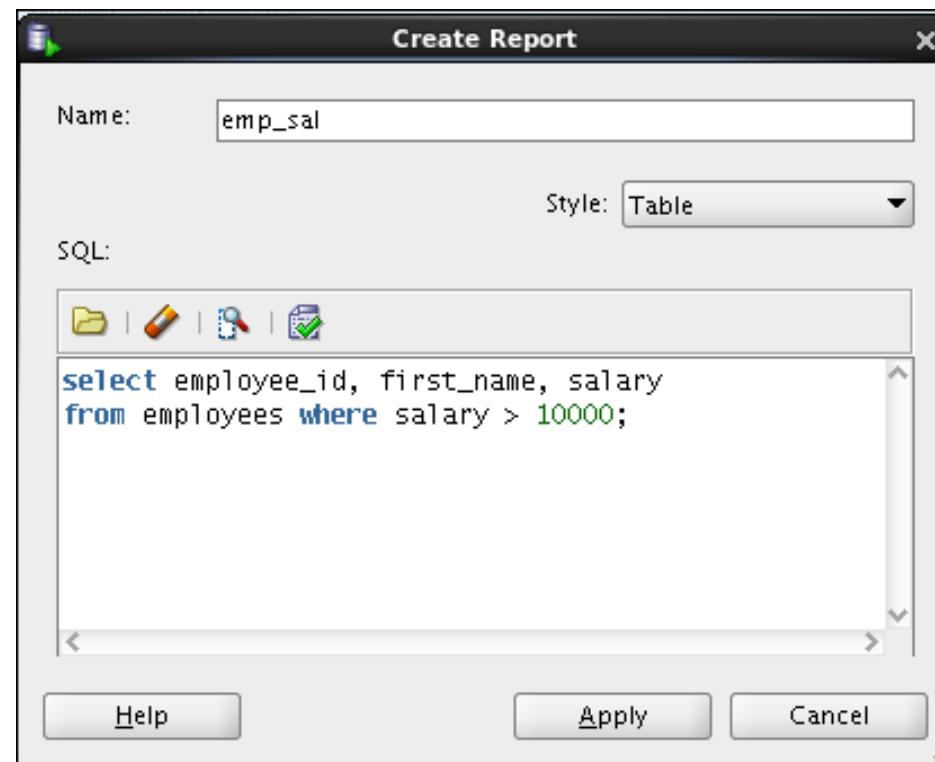
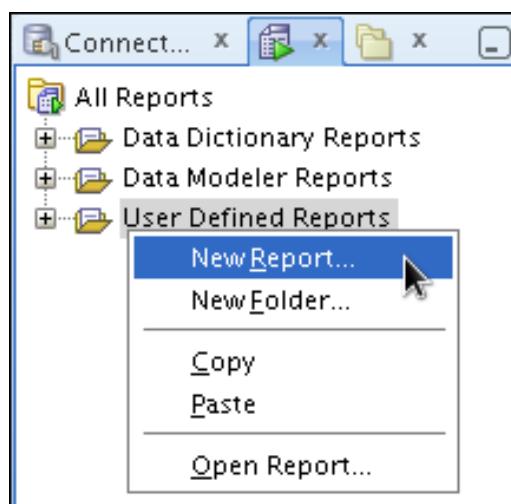
SQL Developer provides a number of predefined reports about the database and its objects.

The screenshot shows the SQL Developer interface with the 'Dependencies' report selected. The left sidebar contains a tree view of predefined reports, with 'Dependencies' highlighted and a red box drawn around it. The main panel displays a table titled 'Dependencies' with the following data:

Owner	Name	Type	Referenced_Owner	Referenced_Name	Referenced_Type
APEX_040100	APEX	PROCEDURE	APEX_040100	WWW_FLOW	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	WWW_FLOW_ISC	PACKAGE
APEX_040100	APEX	PROCEDURE	APEX_040100	WWW_FLOW_SECURITY	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEXWS	PACKAGE	SYS	STANDARD	PACKAGE
APEX_040100	APEX_ADMIN	PROCEDURE	APEX_040100	F	PROCEDURE
APEX_040100	APEX_ADMIN	PROCEDURE	SYS	STANDARD	PACKAGE
APEX_040100	APEX_ADMIN	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	NV	FUNCTION
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOWS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_APPLICATION_GROUPS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_AUTHENTIFICATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_COMPANIES	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_COMPANY_SCHEMAS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_COMPUTATIONS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_ICON_BAR	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_INSTALL_SCRIPTS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_ITEMS	TABLE
APEX_040100	APEX_APPLICATIONS	VIEW	APEX_040100	WWW_FLOW_LANGUAGE_MAP	TABLE

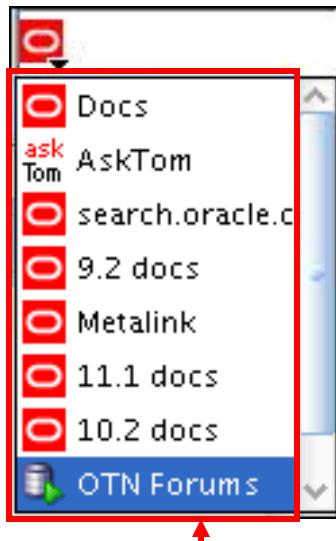
Creating a User-Defined Report

Create and save user-defined reports for repeated use.

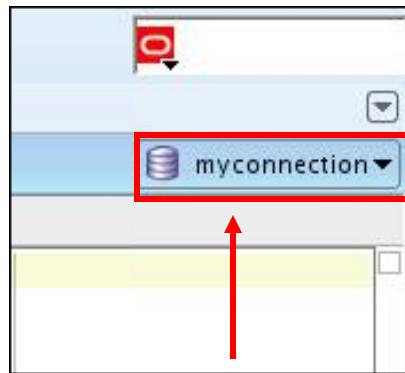


Organize reports in folders.

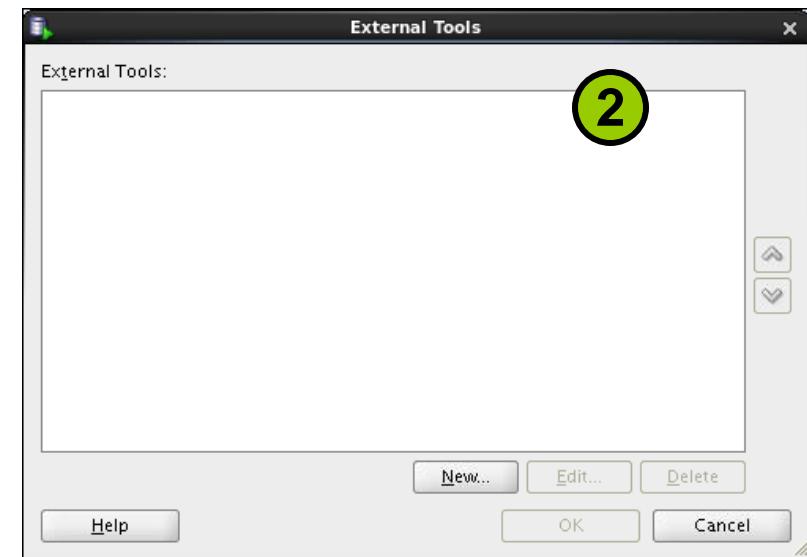
Search Engines and External Tools



Links to popular search engines and discussion forums

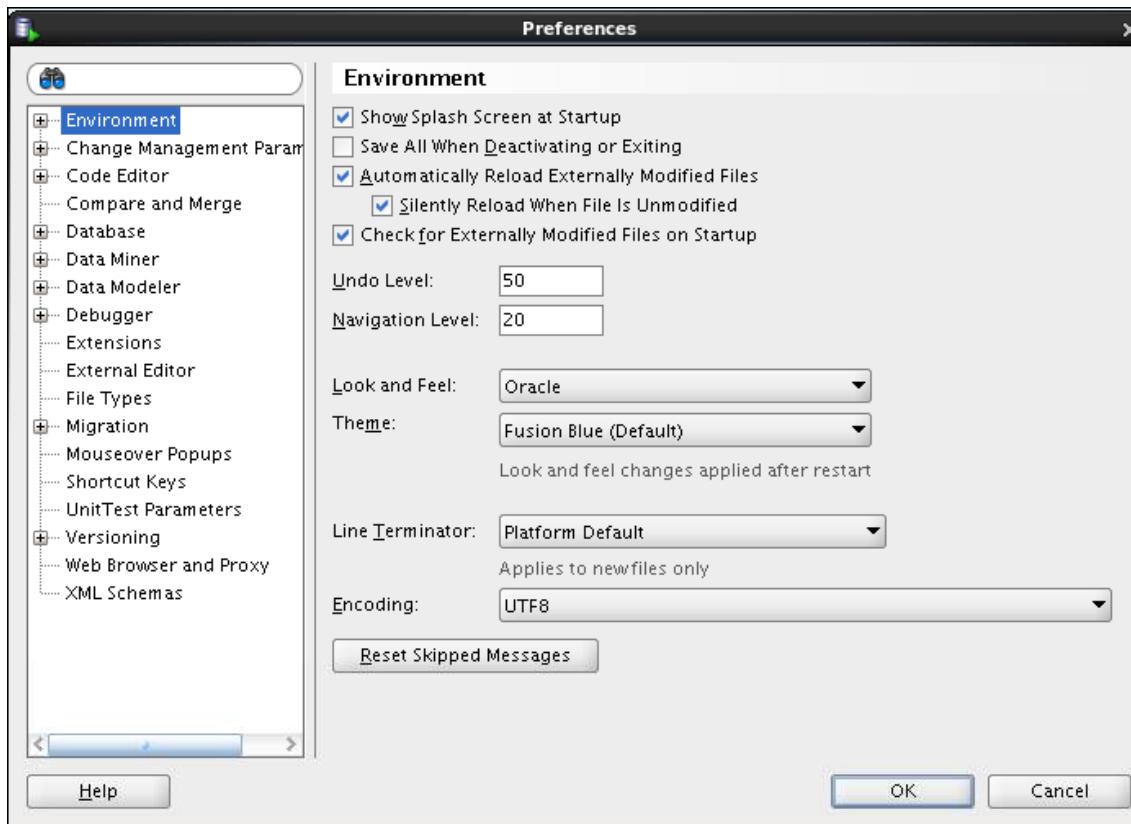


Shortcut to switch between connections

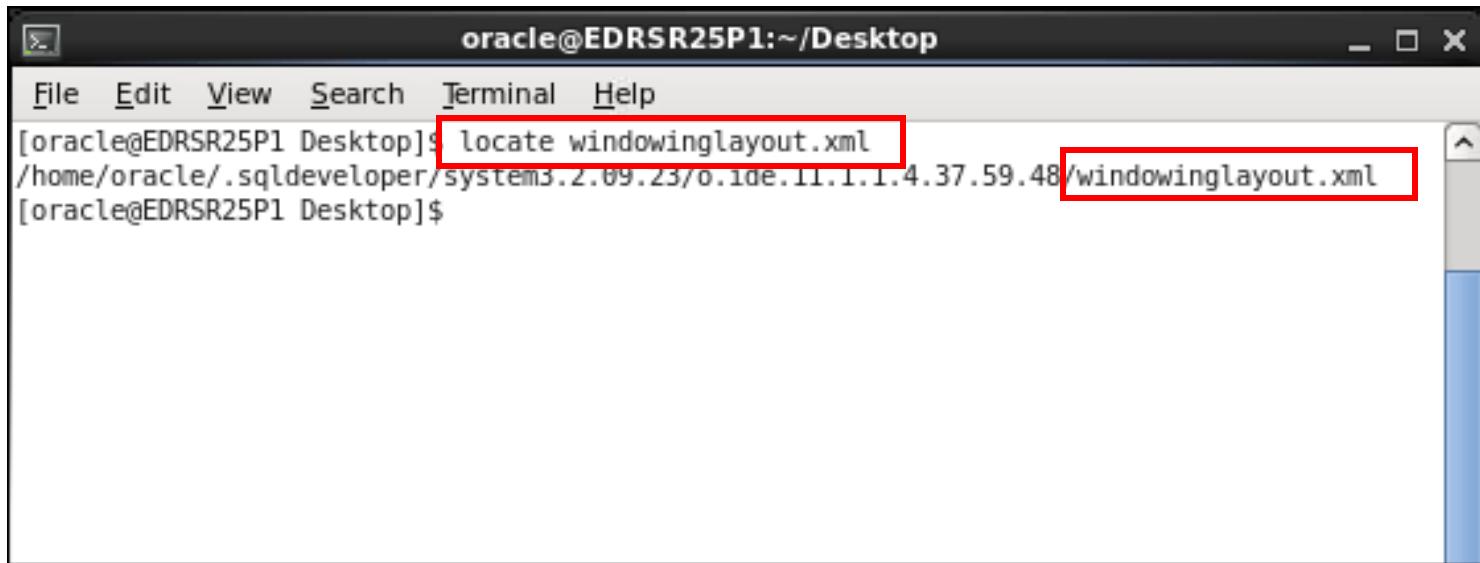


Setting Preferences

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



Resetting the SQL Developer Layout

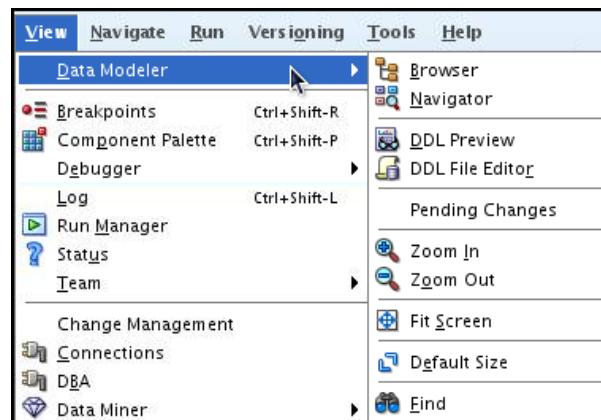
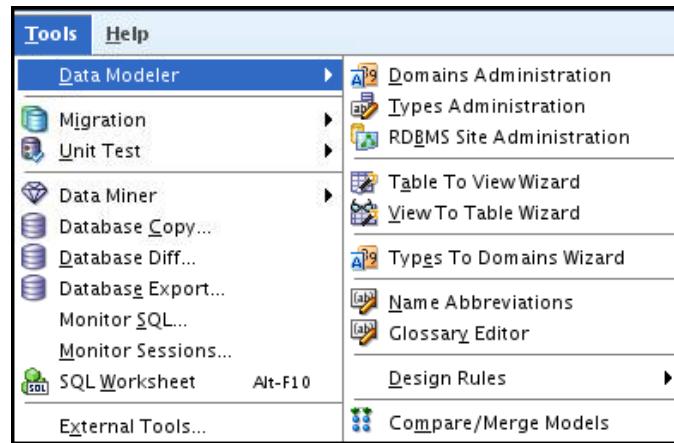
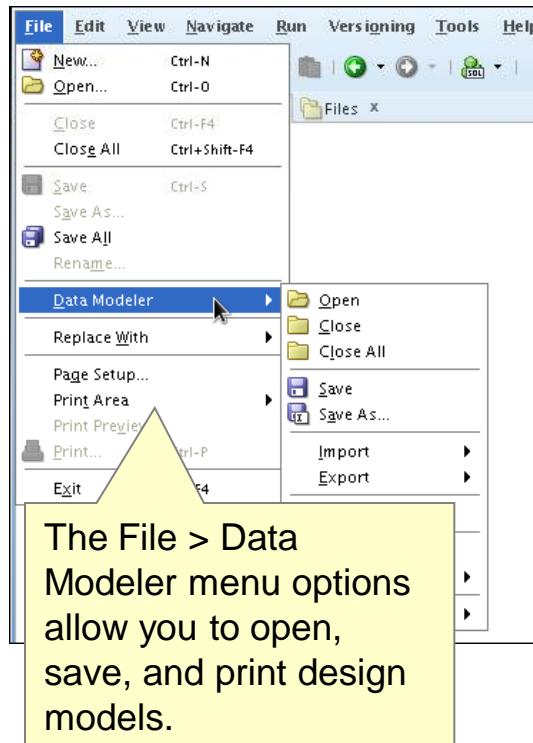


```
oracle@EDRSR25P1:~/Desktop
File Edit View Search Terminal Help
[oracle@EDRSR25P1 Desktop]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system3.2.09.23/0.10e.11.1.1.4.37.59.48/windowinglayout.xml
[oracle@EDRSR25P1 Desktop]$
```

A screenshot of a terminal window titled "oracle@EDRSR25P1:~/Desktop". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command line shows the user running the "locate" command to find files named "windowinglayout.xml". The output of the command is displayed in the main pane, showing the full path to the file located in the ".sqldeveloper" directory. The entire terminal window is enclosed in a red border.

Data Modeler in SQL Developer

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



Summary

In this appendix, you should have learned how to use SQL Developer to do:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports
- Browse the Data Modeling options in SQL Developer



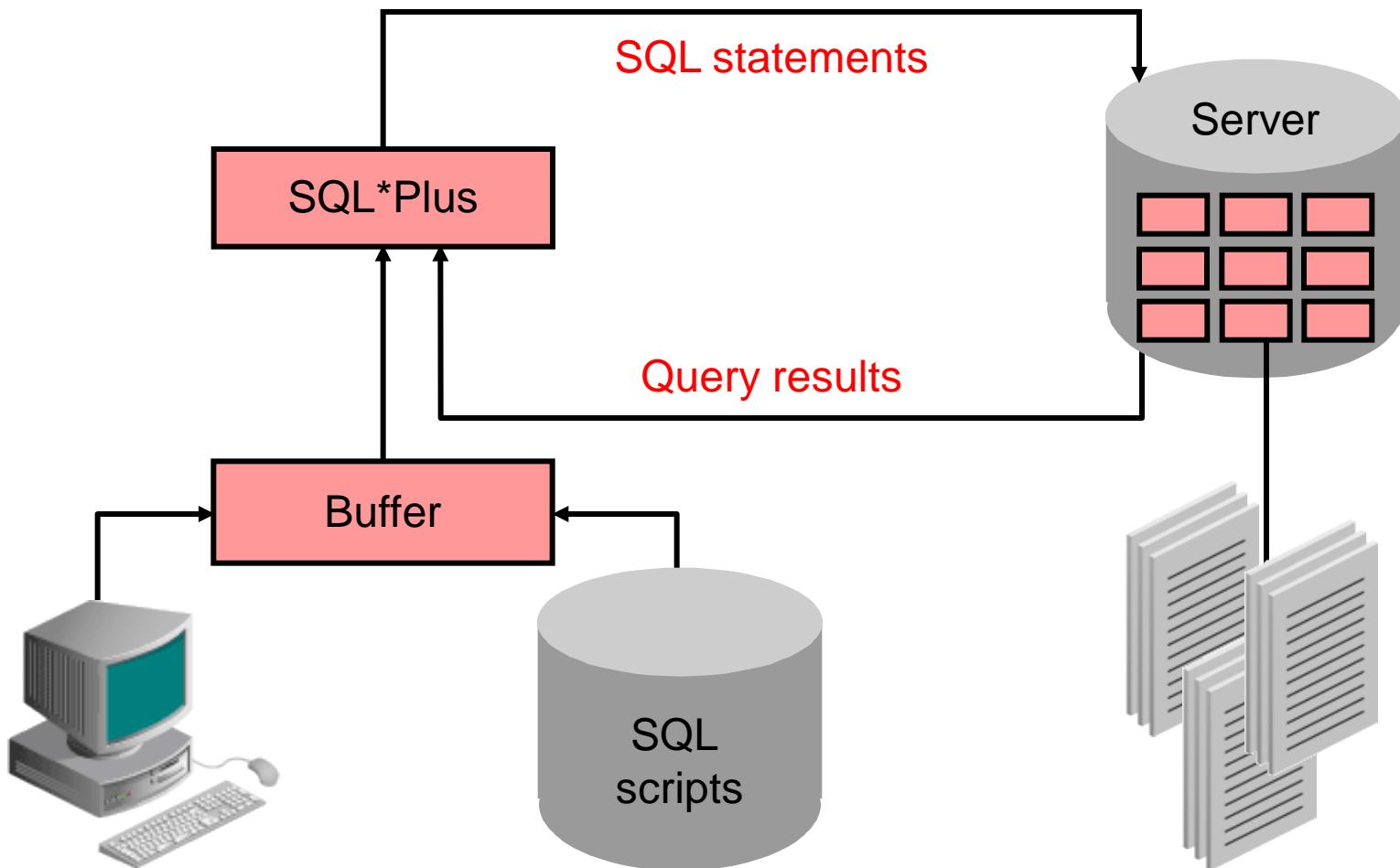
Using SQL*Plus

Objectives

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

- Log in to SQL*Plus
- Edit SQL commands
- Format the output by using SQL*Plus commands
- Interact with script files

SQL and SQL*Plus Interaction



SQL Statements Versus SQL*Plus Commands

SQL

- A language
- ANSI-standard
- Keywords cannot be abbreviated.
- Statements manipulate data and table definitions in the database.

SQL*Plus

- An environment
- Oracle-proprietary
- Keywords can be abbreviated.
- Commands do not allow manipulation of values in the database.



SQL*Plus: Overview

- Log in to SQL*Plus.
- Describe the table structure.
- Edit your SQL statement.
- Execute SQL from SQL*Plus.
- Save SQL statements to files and append SQL statements to files.
- Execute saved files.
- Load commands from the file to buffer to edit.

Logging In to SQL*Plus

A terminal window titled "oracle@EDRSR25P1:~/Desktop". The window shows the following text:
[oracle@EDRSR25P1 Desktop]\$ sqlplus
SQL*Plus: Release 12.1.0.0.2 Beta on Thu Sep 13 02:00:57 2012
Copyright (c) 1982, 2012, Oracle. All rights reserved.
Enter user-name: oral
Enter password:
Last Successful login time: Wed Sep 2012 23:16:13 +00:00
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.0.2 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>

A green circle with the number 1 is positioned over the "Enter user-name" prompt.

```
sqlplus [username[ /password[@database]]]
```

A terminal window titled "oracle@EDRSR25P1:~/Desktop". The window shows the following text:
[oracle@EDRSR25P1 Desktop]\$ sqlplus oral/oral
SQL*Plus: Release 12.1.0.0.2 Beta on Thu Sep 13 02:29:51 2012
Copyright (c) 1982, 2012, Oracle. All rights reserved.
Last Successful login time: Thu Sep 2012 02:01:21 +00:00
Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.0.2 - 64bit Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>

A green circle with the number 2 is positioned over the "Connected to" message.

Displaying the Table Structure

Use the SQL*Plus DESCRIBE command to display the structure of a table:

```
DESC[RIBE] tablename
```

Displaying the Table Structure

```
DESCRIBE departments
```

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

SQL*Plus Editing Commands

- A[PPEND] *text*
- C[HANGE] / *old* / *new*
- C[HANGE] / *text* /
- CL[EAR] BUFF[ER]
- DEL
- DEL *n*
- DEL *m n*

SQL*Plus Editing Commands

- I [NPUT]
- I [NPUT] *text*
- L [IST]
- L [IST] *n*
- L [IST] *m n*
- R [UN]
- *n*
- *n text*
- 0 *text*

Using LIST, n, and APPEND

```
LIST
```

```
1  SELECT last_name  
2* FROM    employees
```

```
1
```

```
1* SELECT last_name
```

```
A , job_id
```

```
1* SELECT last_name, job_id
```

```
LIST
```

```
1  SELECT last_name, job_id  
2* FROM    employees
```



Using the CHANGE Command

```
LIST
```

```
1* SELECT * from employees
```

```
c/employees/departments
```

```
1* SELECT * from departments
```

```
LIST
```

```
1* SELECT * from departments
```

SQL*Plus File Commands

- SAVE *filename*
- GET *filename*
- START *filename*
- @ *filename*
- EDIT *filename*
- SPOOL *filename*
- EXIT

Using the SAVE and START Commands

```
LIST
```

```
1  SELECT last_name, manager_id, department_id  
2* FROM employees
```

```
SAVE my_query
```

```
Created file my_query
```

```
START my_query
```

LAST_NAME	MANAGER_ID	DEPARTMENT_ID
King		90
Kochhar	100	90
...		
107 rows selected.		

SERVEROUTPUT Command

- Use the SET SERVEROUT [PUT] command to control whether to display the output of stored procedures or PL/SQL blocks in SQL*Plus.
- The DBMS_OUTPUT line length limit is increased from 255 bytes to 32767 bytes.
- The default size is now unlimited.
- Resources are not pre-allocated when SERVEROUTPUT is set.
- Because there is no performance penalty, use UNLIMITED unless you want to conserve physical memory.

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNLIMITED}]
[FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NATED]}]
```

Using the SQL*Plus SPOOL Command

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] |  
APP[END]] | OFF | OUT]
```

Option	Description
file_name[.ext]	Spools output to the specified file name
CRE[ATE]	Creates a new file with the name specified
REP[LACE]	Replaces the contents of an existing file. If the file does not exist, REPLACE creates the file.
APP[END]	Adds the contents of the buffer to the end of the file that you specify
OFF	Stops spooling
OUT	Stops spooling and sends the file to your computer's standard (default) printer

Using the AUTOTRACE Command

- It displays a report after the successful execution of SQL DML statements such as SELECT, INSERT, UPDATE, or DELETE.
- The report can now include execution statistics and the query execution path.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]  
[STATISTICS]
```

```
SET AUTOTRACE ON  
-- The AUTOTRACE report includes both the optimizer  
-- execution path and the SQL statement execution  
-- statistics
```

Summary

In this appendix, you should have learned how to use SQL*Plus as an environment to do the following:

- Execute SQL statements
- Edit SQL statements
- Format the output
- Interact with script files

Commonly Used SQL Commands

Objectives

After completing this appendix, you should be able to:

- Execute a basic SELECT statement
- Create, alter, and drop a table using data definition language (DDL) statements
- Insert, update, and delete rows from one or more tables using data manipulation language (DML) statements
- Commit, roll back, and create savepoints by using transaction control statements
- Perform join operations on one or more tables

Basic SELECT Statement

- Use the SELECT statement to:
 - Identify the columns to be displayed
 - Retrieve data from one or more tables, object tables, views, object views, or materialized views
- A SELECT statement is also known as a query because it queries a database.
- Syntax:

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

SELECT Statement

- Select all columns:

```
SELECT *  
FROM job_history;
```

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-01	24-JUL-06	IT_PROG	60
2	101	21-SEP-97	27-OCT-01	AC_ACCOUNT	110
3	101	28-OCT-01	15-MAR-05	AC_MGR	110
4	201	17-FEB-04	19-DEC-07	MK_REP	20
5	114	24-MAR-06	31-DEC-07	ST_CLERK	50
6	122	01-JAN-07	31-DEC-07	ST_CLERK	50
7	200	17-SEP-95	17-JUN-01	AD_ASST	90
8	176	24-MAR-06	31-DEC-06	SA_REP	80
9	176	01-JAN-07	31-DEC-07	SA_MAN	80
10	200	01-JUL-02	31-DEC-06	AC_ACCOUNT	90

- Select specific columns:

```
SELECT manager_id, job_id  
FROM employees;
```

	MANAGER_ID	JOB_ID
1	(null)	AD_PRES
2	100	AD_VP
3	100	AD_VP
4	102	IT_PROG
5	103	IT_PROG
6	103	IT_PROG
7	100	ST_MAN
8	124	ST_CLERK
9	124	ST_CLERK
10	124	ST_CLERK

...

ORACLE

WHERE Clause

- Use the optional WHERE clause to:
 - Filter rows in a query
 - Produce a subset of rows
- Syntax:

```
SELECT * FROM table
[ WHERE condition] ;
```

- Example:

```
SELECT location_id from departments
WHERE department_name = 'Marketing' ;
```

ORDER BY Clause

- Use the optional ORDER BY clause to specify the row order.
- Syntax:

```
SELECT * FROM table
[ WHERE condition ]
[ ORDER BY {<column>|<position>} [ASC|DESC] [, ...] ];
```

- Example:

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

GROUP BY Clause

- Use the optional GROUP BY clause to group columns that have matching values into subsets.
- Each group has no two rows having the same value for the grouping column or columns.
- Syntax:

```
SELECT <column1, column2, ... column_n>
FROM   table
[ WHERE  condition ]
[ GROUP BY <column> [ , ... ] ]
[ ORDER BY <column> [ , ... ] ] ;
```

- Example:

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

Data Definition Language

- DDL statements are used to define, structurally change, and drop schema objects.
- The commonly used DDL statements are:
 - CREATE TABLE, ALTER TABLE, and DROP TABLE
 - GRANT, REVOKE
 - TRUNCATE

CREATE TABLE Statement

- Use the CREATE TABLE statement to create a table in the database.
- Syntax:

```
CREATE TABLE tablename (
  {column-definition | Table-level constraint}
  [ , {column-definition | Table-level constraint} ] * )
```

- Example:

```
CREATE TABLE teach_dept (
  department_id NUMBER(3) PRIMARY KEY,
  department_name VARCHAR2(10));
```

ALTER TABLE Statement

- Use the ALTER TABLE statement to modify the definition of an existing table in the database.
- Example1:

```
ALTER TABLE teach_dept  
ADD location_id NUMBER NOT NULL;
```

- Example 2:

```
ALTER TABLE teach_dept  
MODIFY department_name VARCHAR2( 30 ) NOT NULL;
```

DROP TABLE Statement

- The `DROP TABLE` statement removes the table and all its data from the database.
- Example:

```
DROP TABLE teach_dept;
```

- `DROP TABLE` with the `PURGE` clause drops the table and releases the space that is associated with it.

```
DROP TABLE teach_dept PURGE;
```

- The `CASCADE CONSTRAINTS` clause drops all referential integrity constraints from the table.

```
DROP TABLE teach_dept CASCADE CONSTRAINTS;
```

GRANT Statement

- The GRANT statement assigns the privilege to perform the following operations:
 - Insert or delete data.
 - Create a foreign key reference to the named table or to a subset of columns from a table.
 - Select data, a view, or a subset of columns from a table.
 - Create a trigger on a table.
 - Execute a specified function or procedure.
- Example:

```
GRANT SELECT any table to PUBLIC;
```

Privilege Types

Assign the following privileges by using the GRANT statement:

- ALL PRIVILEGES
- DELETE
- INSERT
- REFERENCES
- SELECT
- UPDATE

REVOKE Statement

- Use the REVOKE statement to remove privileges from a user to perform actions on database objects.
- Revoke a *system privilege* from a user:

```
REVOKE DROP ANY TABLE  
FROM hr;
```

- Revoke a *role* from a user:

```
REVOKE dw_manager  
FROM sh;
```

TRUNCATE TABLE Statement

- Use the TRUNCATE TABLE statement to remove all the rows from a table.
- Example:

```
TRUNCATE TABLE employees_demo;
```

- By default, Oracle Database performs the following tasks:
 - De-allocates space used by the removed rows
 - Sets the NEXT storage parameter to the size of the last extent removed from the segment by the truncation process

Data Manipulation Language

- DML statements query or manipulate data in the existing schema objects.
- A DML statement is executed when:
 - New rows are added to a table by using the `INSERT` statement
 - Existing rows in a table are modified using the `UPDATE` statement
 - Existing rows are deleted from a table by using the `DELETE` statement
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

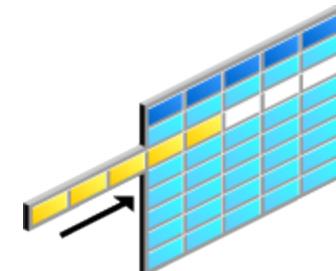
INSERT Statement

- Use the INSERT statement to add new rows to a table.
- Syntax:

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

- Example:

```
INSERT INTO departments  
VALUES      (200, 'Development', 104, 1400);  
1 rows inserted.
```



UPDATE Statement Syntax

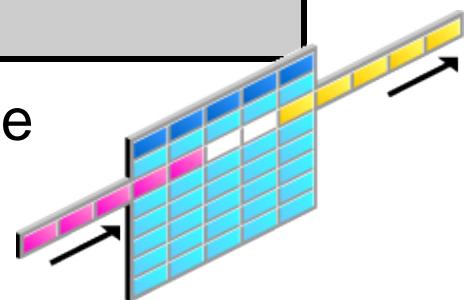
- Use the UPDATE statement to modify the existing rows in a table.
- Update more than one row at a time (if required).

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

- Example:

```
UPDATE      copy_emp
SET
22 rows updated
```

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



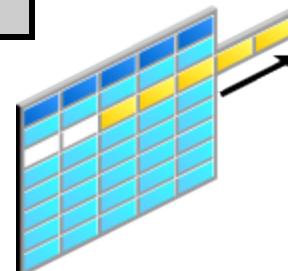
DELETE Statement

- Use the DELETE statement to delete the existing rows from a table.
- Syntax:

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

- Write the DELETE statement using the WHERE clause to delete specific rows from a table.

```
DELETE FROM departments  
WHERE department_name = 'Finance';  
1 rows deleted
```



Transaction Control Statements

- Transaction control statements are used to manage the changes made by DML statements.
- The DML statements are grouped into transactions.
- Transaction control statements include:
 - COMMIT
 - ROLLBACK
 - SAVEPOINT

COMMIT Statement

- Use the COMMIT statement to:
 - Permanently save the changes made to the database during the current transaction
 - Erase all savepoints in the transaction
 - Release transaction locks
- Example:

```
INSERT INTO departments  
VALUES      ( 201 , 'Engineering' , 106 , 1400 ) ;  
COMMIT ;
```

```
1 rows inserted.  
committed.
```

ROLLBACK Statement

- Use the ROLLBACK statement to undo changes made to the database during the current transaction.
- Use the TO SAVEPOINT clause to undo a part of the transaction after the savepoint.
- Example:

```
UPDATE          employees
SET            salary = 7000
WHERE          last_name = 'Ernst';
SAVEPOINT      Ernst_sal;

UPDATE          employees
SET            salary = 12000
WHERE          last_name = 'Mourgos';

ROLLBACK TO SAVEPOINT Ernst_sal;
```



SAVEPOINT Statement

- Use the SAVEPOINT statement to name and mark the current point in the processing of a transaction.
- Specify a name to each savepoint.
- Use distinct savepoint names within a transaction to avoid overriding.
- Syntax:

```
SAVEPOINT savepoint;
```

Joins

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

```
SELECT      table1.column, table2.column  
FROM        table1, table2  
WHERE       table1.column1 = table2.column2;
```

- Write the join condition in the WHERE clause.
- Prefix the column name with the table name when the same column name appears in more than one table.

Types of Joins

- Natural join
- Equijoin
- Nonequijoin
- Outer join
- Self-join
- Cross join

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use table aliases instead of full table name prefixes.
- Table aliases give a table a shorter name.
 - This keeps SQL code smaller and uses less memory.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

Natural Join

- The NATURAL JOIN clause is based on all the columns in the two tables that have the same name.
- It selects rows from tables that have the same names and data values of columns.
- Example:

```
SELECT country_id, location_id, country_name, city  
FROM countries NATURAL JOIN locations;
```

	COUNTRY_ID	LOCATION_ID	COUNTRY_NAME	CITY
1	US	1400	United States of America	Southlake
2	US	1500	United States of America	San Francisco
3	US	1700	United States of America	Seattle
4	CA	1800	Canada	Toronto
5	UK	2500	United Kingdom	Oxford

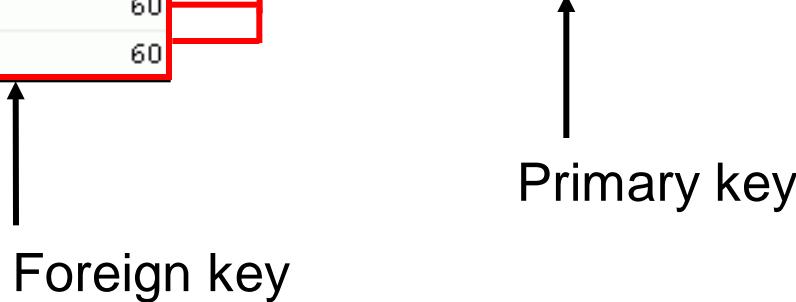
Equijoins

EMPLOYEES

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

DEPARTMENTS

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



Retrieving Records with Equijoins

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

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	Whalen	10	10	1700
2	Hartstein	20	20	1800
3	Fay	20	20	1800
4	Vargas	50	50	1500
5	Matos	50	50	1500
6	Davies	50	50	1500
7	Rajs	50	50	1500
8	Mourgos	50	50	1500
9	Hunold	60	60	1400
10	Ernst	60	60	1400
11	Lorentz	60	60	1400
...				

Additional Search Conditions Using the AND and WHERE Operators

```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
AND d.department_id IN (20, 50);
```



	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco



```
SELECT d.department_id, d.department_name, l.city
FROM departments d JOIN locations l
ON d.location_id = l.location_id
WHERE d.department_id IN (20, 50);
```

Retrieving Records with NonequiJoins

```
SELECT e.last_name, e.salary, j.grade_level  
FROM employees e JOIN job_grades j  
ON e.salary  
    BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Fay	6000	C

...

ORACLE®

Retrieving Records by Using the USING Clause

- You can use the USING clause to match only one column when more than one column matches.
- You cannot specify this clause with a NATURAL join.
- Do not qualify the column name with a table name or table alias.
- Example:

```
SELECT country_id, country_name, location_id, city
FROM   countries JOIN locations
USING (country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	US	United States of America	1400	Southlake
2	US	United States of America	1500	South San Francisco
3	US	United States of America	1700	Seattle
4	CA	Canada	1800	Toronto
5	UK	United Kingdom	2500	Oxford



Retrieving Records by Using the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The ON clause makes code easy to understand.

```
SELECT e.employee_id, e.last_name, j.department_id,  
FROM   employees e JOIN job_history j  
ON     (e.employee_id = j.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	101	Kochhar	110
2	101	Kochhar	110
3	102	De Haan	60
4	176	Taylor	80
5	176	Taylor	80
6	200	Whalen	90
7	200	Whalen	90
8	201	Hartstein	20



Left Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the left table is called a LEFT OUTER JOIN.
- Example:

```
SELECT c.country_id, c.country_name, l.location_id, l.city
FROM   countries c LEFT OUTER JOIN locations l
ON    (c.country_id = l.country_id) ;
```

	COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	CITY
1	CA	Canada	1800	Toronto
2	DE	Germany	(null)	(null)
3	UK	United Kingdom	2500	Oxford
4	US	United States of America	1400	Southlake
5	US	United States of America	1500	South San Francisco
6	US	United States of America	1700	Seattle

Right Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from the right table is called a RIGHT OUTER JOIN.
- Example:

```
SELECT e.last_name, d.department_id, d.department_name  
FROM   employees e RIGHT OUTER JOIN departments d  
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
...			

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

Full Outer Join

- A join between two tables that returns all matched rows, as well as the unmatched rows from both tables is called a FULL OUTER JOIN.
- Example:

```
SELECT e.last_name, d.department_id, d.manager_id,
       d.department_name
  FROM employees e FULL OUTER JOIN departments d
 WHERE (e.manager_id = d.manager_id) ;
```

	LAST_NAME	DEPARTMENT_ID	MANAGER_ID	DEPARTMENT_NAME
1	King	(null)	(null)	(null)
2	Kochhar	90	100	Executive
3	De Haan	90	100	Executive
4	Hunold	(null)	(null)	(null)

...

19	Higgins	(null)	(null)	(null)
20	Gietz	110	205	Accounting
21	(null)	190	(null)	Contracting
22	(null)	10	200	Administration

Self-Join: Example

```
SELECT worker.last_name || ' works for '  
      || manager.last_name  
FROM   employees worker JOIN employees manager  
ON worker.manager_id = manager.employee_id  
ORDER BY worker.last_name;
```

WORKER.LAST_NAME 'WORKSFOR' MANAGER.LAST_NAME
1 Abel works for Zlotkey
2 Davies works for Mourgos
3 De Haan works for King
4 Ernst works for Hunold
5 Fay works for Hartstein
6 Gietz works for Higgins
7 Grant works for Zlotkey
8 Hartstein works for King
9 Higgins works for Kochhar

Cross Join

- A CROSS JOIN is a JOIN operation that produces the Cartesian product of two tables.
- Example:

```
SELECT department_name, city  
FROM department CROSS JOIN location;
```

	DEPARTMENT_NAME	CITY
1	Administration	Oxford
2	Administration	Seattle
3	Administration	South San Francisco
4	Administration	Southlake
5	Administration	Toronto
6	Marketing	Oxford
7	Marketing	Seattle
8	Marketing	South San Francisco
9	Marketing	Southlake
10	Marketing	Toronto

...

ORACLE®

Summary

In this appendix, you should have learned how to use:

- The SELECT statement to retrieve rows from one or more tables
- DDL statements to alter the structure of objects
- DML statements to manipulate data in the existing schema objects
- Transaction control statements to manage the changes made by DML statements
- Joins to display data from multiple tables